**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**
**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**
**імені ІГОРЯ СІКОРСЬКОГО»**

**Факультет прикладної математики**

**Кафедра програмного забезпечення комп'ютерних систем**

«До захисту допущено»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2020 р.

# Дипломний проєкт

**на здобуття ступеня бакалавра**

**за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних та інформаційно-пошукових систем»**

**спеціальності 121 Інженерія програмного забезпечення**

**на тему: «Соціально-аналітичний веб-сервіс для побудови рекомендацій»**

Виконала:
студент IV курсу, групи КП-62
Пінтак Дмитро Львович _____

Керівник:
Старший викладач кафедри ПЗКС, к.т.н.
Рибачок Наталія Антонівна _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович _____

Рецензент:
к.т.н., с.н.с. МННЦ ІТС НАНУ та МОНУ
Савченко Євгенія Анатоліївна _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

**Національний технічний університет України**

**«Київський політехнічний інститут імені Ігоря Сікорського»**

**Факультет прикладної математики**

**Кафедра програмного забезпечення комп'ютерних систем**

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення комп'ютерних та інформаційно-пошукових систем»

«ЗАТВЕРДЖУЮ»

Науковий керівник кафедри

_____ Іван ДИЧКА

«___»_____2019 р.

# ЗАВДАННЯ

**на дипломний проєкт студенту**

Пінтаку Дмитру Львовичу

1. Тема проєкту «Соціально-аналітичний веб-сервіс для побудови рекомендацій», керівник проєкту Рибачок Наталія Антонівна, ст. викладач, к.т.н., затверджені наказом по університету від «25» травня 2020 р. №1181-с

2. Термін подання студентом проєкту «16» червня 2020 р.

3. Вихідні дані до проєкту: див. Технічне завдання.

4. Зміст пояснювальної записки:
   - огляд наявних рішень
   - обґрунтування вибору засобів реалізації;
   - опис розробленої системи;
   - аналіз розробленої системи.

5. Перелік обов'язкового графічного матеріалу:
   - діаграма варіантів використання (креслення);
   - діаграма компонентів (креслення);
   - інтерфейс користувача (плакат);

– архітектура системи (плакат).

6. Консультанти розділів проєкту

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| Нормоконтроль | Онай М.В., доцент | | |

7. Дата видачі завдання «31» жовтня 2019 р.

Календарний план

| № з/п | Назва етапів виконання дипломного проєкту | Термін виконання етапів проєкту | Примітка |
|---|---|---|---|
| 1. | Вивчення літератури за тематикою проєкту | 10.11.2019 | |
| 2. | Розроблення та узгодження технічного завдання | 27.11.2019 | |
| 3. | Розроблення структури веб-сервісу | 14.12.2019 | |
| 4. | Підготовка матеріалів першого розділу дипломного проєкту | 27.12.2019 | |
| 5. | Розроблення дизайну сторінок та графічних елементів | 02.02.2020 | |
| 6. | Підготовка матеріалів другого розділу дипломного проєкту | 19.02.2020 | |
| 7. | Програмна реалізація веб-сервісу | 12.03.2020 | |
| 8. | Тестування веб-сервісу | 20.03.2020 | |
| 9. | Підготовка матеріалів третього розділу дипломного проєкту | 04.04.2020 | |
| 10. | Підготовка матеріалів четвертого розділу дипломного проєкту | 24.04.2020 | |
| 11. | Підготовка графічної частини дипломного проєкту | 12.05.2020 | |
| 12. | Оформлення документації дипломного проєкту | 24.05.2020 | |

Студент Дмитро ПІНТАК

Керівник проєкту Наталя РИБАЧОК

# АНОТАЦІЯ

Даний дипломний проєкт присвячений створенню веб-сервісу для надання рекомендацій на основі соціальних зв’язків клієнтів, використовуючи дані з соціальних мереж. Вибір теми обумовлений відсутністю наявних рішень, які при побудові рекомендацій використовують соціальні зв’язки клієнтів.

Під час роботи виконано порівняльний аналіз наявних рішень для побудови рекомендацій, проаналізовано методи побудови рекомендацій, обґрунтовано вибір технологій та допоміжних бібліотек серверної та клієнтської частин для реалізації даного програмного продукту. Розроблений веб-сервіс надає можливість користувачам надавати персоналізовані рекомендації клієнтам, основуючись на соціальних зв’язках між ними, які отримуються з соціальних мереж. Процес авторизації клієнтів та збір інформації про них відбувається в автоматичному режимі, після їх переадресації з додатків користувачів на сервер системи, з подальшою переадресацією на соціальну мережу. Більшість послуг користувачам надається за допомогою HTTP API або використовуючи веб-інтерфейс.

У даному проєкті розроблено: архітектуру серверної та клієнтської частини веб-сервісу, алгоритм збору інформації з соціальних мереж, алгоритм побудови зв’язків між клієнтами та алгоритм побудови рекомендацій, графічні елементи та дизайн веб-сторінок.

# ABSTRACT

This diploma project is dedicated to the creation of a web service for providing recommendations based on social connections of clients, using data from social networks. The choice of topic is due to the lack of existing solutions that use the social connections of clients when building recommendations.

During the work the comparative analysis of the available decisions for construction of recommendations is executed, the methods of construction of recommendations are analyzed, the choice of technologies and auxiliary libraries of server and client parts for realization of the given software product is substantiated. The developed web service allows users to provide personalized recommendations to customers, based on the social connections between them, which are obtained from social networks. The process of authorizing clients and collecting information about them is automatic, after their redirection from user applications to the system server, followed by redirection to the social network. Most services are provided to users using the HTTP API or using a web interface.

This project has developed: the architecture of the server and client part of the web service, the algorithm for collecting information from social networks, the algorithm for building relationships between clients and the algorithm for building recommendations, graphics and design of web pages.

ДП.045440-01-90 Соціально-аналітичний веб-сервіс для побудови рекомендацій. Відомість проєкту

| Позначення | Найменування | Кіл-ть | Примітка |
|---|---|---|---|
| | Документація проєкту | | |
| | | | |
| ДП.045440-02-91 | Соціально-аналітичний | 5 | |
| | веб-сервіс для побудови | | |
| | рекомендацій. Технічне | | |
| | завдання | | |
| | | | |
| ДП.045440-03-81 | Соціально-аналітичний | 50 | |
| | веб-сервіс для побудови | | |
| | рекомендацій. | | |
| | Пояснювальна записка | | |
| | | | |
| ДП.045440-04-51 | Соціально-аналітичний | 4 | |
| | веб-сервіс для побудови | | |
| | рекомендацій. Програма | | |
| | та методика тестування | | |
| | | | |
| ДП.045440-05-34 | Соціально-аналітичний | 9 | |
| | веб-сервіс для побудови | | |
| | рекомендацій. | | |
| | Керівництво користувача | | |
| | | | |
| ДП.045440-06-99 | Соціально-аналітичний | 1 | |
| | веб-сервіс для побудови | | |
| | рекомендацій. Діаграма | | |
| | варіантів використання. | | |
| | UML-діаграма | | |
| | прецедентів | | |
| | | | |
| | | | |
| | | | |
| | | | |

| Позначення | Найменування | Кіл-ть | Примітка |
|---|---|---|---|
| | | | |
| ДП.045440-07-99 | Соціально-аналітичний | 1 | |
| | веб-сервіс для побудови | | |
| | рекомендацій. Діаграма | | |
| | компонентів. | | |
| | UML-діаграма | | |
| | | | |
| ДП.045440-08-98 | Соціально-аналітичний | 1 | |
| | веб-сервіс для побудови | | |
| | рекомендацій. | | |
| | Компакт-диск | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

**Факультет прикладної математики**

**Кафедра програмного забезпечення комп'ютерних систем**

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2019 р.

# СОЦІАЛЬНО-АНАЛІТИЧНИЙ ВЕБ-СЕРВІС ДЛЯ ПОБУДОВИ РЕКОМЕНДАЦІЙ

## Технічне завдання

ДП.045440-02-91

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Наталя РИБАЧОК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Дмитро ПІНТАК

2019

**ЗМІСТ**

## 1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

**Назва розробки**: Соціально-аналітичний веб-сервіс для побудови рекомендацій.

**Галузь застосування**: інформаційні технології.

## 2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проєктування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

## 3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розроблюване програмне забезпечення призначене для надавання рекомендацій, основуючись на соціальних зв'язках клієнтів та подальшого аналізу даних про дії клієнтів.

## 4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

- соціально-аналітичний веб-сервіс має бути представлений у вигляді вебклієнту та серверу з HTTP API для розробників;
- авторизація клієнта через соціальну мережу;
- отримання персоналізованих клієнтських рекомендацій у вигляді виборки товарів;
- завантаження дій клієнтів;
- перегляд останніх дій клієнтів та популярних товарів.

Розробку серверної частини виконати на платформі Node.js з використанням фреймворку NestJS, клієнтської – з використанням бібліотек ReactJS та Redux.

## 5. ВИМОГИ ДО ПРОЄКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проєкту повинна бути розроблена наступна документація:

- пояснювальна записка;
- програма та методика тестування;
- керівництво користувача;
- креслення:
  - «Діаграма компонентів»;
  - «Діаграма варіантів використання».

## 6. ЕТАПИ ПРОЄКТУВАННЯ

Вивчення літератури за тематикою проєкту……10.11.2019

Розроблення та узгодження технічного завдання……....27.11.2019

Розроблення структури веб-сервісу…….....14.12.2019

Підготовка матеріалів першого розділу проєкту…….27.12.2019

Розроблення дизайну сторінок та графічних елементів……..02.02.2020

Підготовка матеріалів другого розділу проєкту……...19.02.2020

Програмна реалізація веб-сервісу…….....12.03.2020

Тестування веб-сервісу...……....20.03.2020

Підготовка матеріалів третього розділу проєкту…….04.04.2020

Підготовка матеріалів четвертого розділу проєкту…….24.04.2020

Підготовка матеріалів графічної частини проєкту…….....12.05.2020

Оформлення технічної документації проєкту……..…24.05.2020

## 7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

**Факультет прикладної математики**

**Кафедра програмного забезпечення комп'ютерних систем**

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2020 р.

# СОЦІАЛЬНО-АНАЛІТИЧНИЙ ВЕБ-СЕРВІС ДЛЯ ПОБУДОВИ РЕКОМЕНДАЦІЙ

**Пояснювальна записка**

ДП.045440-03-81

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Наталя РИБАЧОК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Дмитро ПІНТАК

2020

**ЗМІСТ**

## СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

***Стрімінг*** – потокове отримання даних, зазвичай використовується в контексті сервісів мультимедіа.

***ПЗ*** – програмне забезпечення.

***Фреймворк*** – набір готових компонентів, які використовуються для подальшої композиції та модифікації.

***СУБД*** – система управління базами даних, програмне забезпечення що використовується задля керування даними.

***Транзакція*** – цільна одиниця виконаної роботи, що отримує або модифікує дані.

***ACID*** – це набір властивостей, що гарантують надійну роботу транзакцій бази даних: атомарність, узгодженість, ізольованість, довговічність.

***Хешування*** – процес перетворення даних за допомогою асиметричного алгоритма шифрування.

***HTTP*** – протокол, що використовується при передачі даних в комп’ютерних мережах

***API*** – прикладний програмний інтерфейс, сукупність правил, що вможливлюють взаємодію між програмами.

***AI*** – штучний інтелект.

***Real-time*** – прикметник, що описує дані, які оновлюються одразу після здійснення дій що їх змінюють.

***Інтеграція*** – процес налаштування взаємодії між двома системами, або підпрограмами.

***ОС*** – операційна система.

***IDE*** – інтегроване середовище розробки.

***Хост*** – комп’ютер, що містить ресурс та надає до нього доступ.

***Вайтпейпер*** – маркетинговий інтрумент, що описує стратегію компанії.

***Автентифікація*** – процедура встановлення принадлежності інформації до наданого ідентифікатора.

***Авторизація*** – керування рівнями та засобами доступу до певного захищеного ресурсу.

***Ідентифікація*** – процедура розпізнавання користувача в системі за наданим ідентифікатором.

***Граф*** – сукупність об'єктів із зв'язками між ними.

***Master-Slave*** – модель комунікації між двома програмами, що полягає в асиметричному керуванні.

***Булеве значення*** – значення яке дорівнює 1 або 0.

***Веб-сторінка*** – інформаційний ресурс, доступний в мережі інтернет.

***DOM*** – об'єктна модель документа, специфікація прикладного програмного інтерфейсу для роботи з документами.

***Браузер*** – програмне забезпечення, що надає можливість переглядати веб-сторінки.

***Рендеринг*** – процес отримання візуально представлення з моделі.

***Кросплатфоменість*** – властивісь програмного забеспечення, що підримуються декількома ОС.

***Шифрування*** – криптографічне перетворення даних, яке виконується у посимвольній послідовності.

***Масштабованість*** – властивість програмного забеспечення, що може бути легко розширене.

***Бібліотека*** – збірка готових до використання підпрограм або обьєктів для вирішення близьких за тематикою задач.

***Плагін*** – додаток-підпрограма що динамічно підключається до основної програми.

***Веб-сайт*** – набір веб-сторінок.

***JSON*** – формат зберігання та обміну даних, що базується на мові JavaScript.

***Обфускація*** – процес перетворення вихідного коду на тотожну послідовність символів, яку буде неможливо використати для подальшої розробки.

***Back-end*** – це серверна частина проєкту, яка виконує бізнес логіку та є пластом доступу до даних.

***Front-end*** – це клієнтська частина проєкту, абстракція над пластом *Back-end*, яка надає доступ до розташованого за ним функціоналу через інтерфейс, зручний для користувача.

# ВСТУП

Рекомендаційна система – це одна із найважливіших частин будь-яких торгівельних площадок, інтернет-магазинів, сервісів для стрімінгу музикальних композицій, а також незліченної кількості комерційних проєктів.

Розквіт інтернету все більше сприяє появі нових типів веб-сайтів новин з порталами електронної комерції. Інтернет-сайти новин надають конкретну інформацію на такі теми, як стиль життя, новини моди та різноманітні інші види діяльності. Надання онлайн-рекомендацій щодо діяльності, пов’язаної з веб-сайтами новин, може потенційно залучити більше користувачів та створити більше переваг. Такі онлайн-рекомендації є важливою тенденцією в Інтернеті.

Система рекомендацій допомагає користувачам знаходити переконливий вміст у великій корпорації. Наприклад, Google Play Store надає мільйони програм, а YouTube надає мільярди відео. Щодня додається більше додатків та відео. Як варіант, для доступу до вмісту можна використовувати пошук. Однак система рекомендацій може відображати елементи, які користувачі, можливо, не думали шукати самостійно, але мають бажання переглянути.

Жодний сучасний успішний проєкт не може існувати без гнучкої та точно налаштованої системи рекомендацій. Дослідження довели що вправно налаштоване рекомендаційне ПЗ може суттєво збільшити прибутки компанії. Наприклад Amazon отримує 35% виручки тільки за допомогою рекомендаційного двигуна.

У той же час, через те, що популярність вищеописаних сервісів ще не досягла свого піку, їх автори постійно експериментують із різними підходами до процесу надання рекомендацій, кожен з яких містить свої недоліки.

Одним із таких недоліків вважають складність інтеграції подібних сервісів у невеликі проєкти, недостача даних для надання точної рекомендації та складність налаштування. Метою даного дипломного проєкту є розробка системи для побудови рекомендацій на основі соціальних даних.

# 1. ОГЛЯД НАЯВНИХ РІШЕНЬ

## 1.1. Огляд методів надання рекомендацій

На сьогодні існує безліч різноманітних рекомендаційних систем і методів для надання рекомендацій, але три найпопулярніші типи є:

- спільна фільтрація;
- фільтрування на основі вмісту;
- гібридні системи рекомендацій.

### *1.1.1. Спільна фільтрація*

Метод спільної фільтрації ґрунтуються на зборі та аналізі великого обсягу інформації про поведінку, діяльність чи вподобання користувачів та прогнозуванні того, що сподобається користувачам, виходячи з подібності з іншими користувачами. Основна перевага підходу спільної фільтрації полягає в тому, що він не покладається на машинно проаналізований вміст, і тому він здатний точно рекомендувати складні елементи, такі як фільми, не вимагаючи «розуміння» самого елемента. Більшість алгоритмів використовувались для вимірювання подібності користувачів або подібності елементів у системах рекомендацій.

### *1.1.2. Фільтрування на основі вмісту*

Метод фільтрування на основі вмісту ґрунтуються на описі елемента та профілю налаштування користувача. Ключові слова використовуються для опису товарів, при створенні профіля користувача, вказуються типи товарів, які йому подобаються. Іншими словами, ці алгоритми намагаються рекомендувати товари, схожі на ті, які користувач вподобав у минулому. Зокрема, різні товари-кандидати порівнюються з товарами, які раніше оцінив користувач, і рекомендуються найбільш відповідні.

### *1.1.3. Гібридні системи рекомендацій*

Недавні дослідження показали, що гібридний метод, що поєднує спільну фільтрацію та фільтрування на основі вмісту, може бути ефективнішим у деяких випадках. Гібридні підходи можуть бути

реалізовані декількома способами, роблячи передбачення на основі вмісту та спільної фільтрації окремо, а потім комбінуючи результати, додаючи вмістові можливості до підходу на основі спільної фільтрації (і навпаки) або об'єднуючи підходи в один гібридний метод. Кілька досліджень емпірично порівнюють ефективність роботи гібридного метода з оригінальними, і демонструють, що гібридні методи можуть дати більш точні рекомендації, ніж оригінальні. Ці методи також можуть бути використані для подолання деяких найпоширеніших проблем у рекомендаційних системах, таких як холодний старт (без існуючих даних).

Netflix – наглядний приклад гібридної системи. Вони дають рекомендації, порівнюючи історію переглядів та пошуку схожих користувачів (тобто спільну фільтрацію), а також пропонуючи фільми, які поділяють характеристики з фільмами, які користувач високо оцінив (фільтрування на основі вмісту).

## 1.2. Огляд популярних реалізацій надання рекомендацій для методу спільної фільтрації

Існує дві реалізації побудови системи спільної фільтрації: матрична факторизація та Item2vec.

### *1.2.1. Матрична факторизація*

Класичний реалізація – матрична факторизація. Мета – заповнити невідомі в матриці взаємодії між користувачами (назвемо це $R$). Задано дві матриці $U$ та $I$, такі що добуток $U$ та $I$ дорівнює $R$. Використовуючи знайдений добуток матимемо значення для невідомих значень $R$, які потім можуть бути використані для створення рекомендацій.

По-перше, необхідно зіставити кожного користувача та елемент у вектор з розмірами $M$ і $N$ відповідно. Це означає, що потрібно визначити значення користувачів та елементів, як правило, вкладені в класифікацію (оскільки дані поняття вкладені у векторний простір). Оскільки ще не відомо значень цих векторів, то реалізація починається з

випадкової ініціалізації. Тоді для кожної взаємодії між елементами користувача (*u, x*) об'єднуються як значення користувача *u*, так і елемент *x*, щоб дати один вектор. Оскільки відомо значення взаємодії між користувачем, то з’являється можливість отримати приблизні значення за допомогою системи. Потім мережа використовуватиме зворотне перетворення для самостійного знаходження рекомендацій. Таким чином, система визначить найкращий спосіб представлення користувачів та елементів, і буде доцільно завантажувати взаємодії, яких вона раніше не бачила, подаючи їй приклади.

### *1.2.2. Item2Vec*

Item2vec пропонує, щоб товари що відповідають один одному можна було знайти використовуючи алгоритм схожий на Word2vec, тобто використовуючи векторизацію, але при цьому враховувати всі дані про товар при формуванні вектора. В алгоритмі використовується контекстна інформація, з якої випливає, що товари, придбані за аналогічних обставин, є порівнянними.

Цей підхід безпосередньо не залучає користувачів, але враховує їх вподобання на момент надання рекомендацій. Тим не менш, це може бути корисно, якщо наша мета – показати користувачеві альтернативи певному предмету, який вони обрали («ви купили цей телевізор, можливо, вам сподобається і інший»). Основною проблемою є необхідність великої кількості даних, щоб створити доцільні рекомендації. У вайтпейпері Item2vec було використано два набори даних: один із 9 мільйонами взаємодій, 732 тис. користувачів та 49 тис. елементів та інший із 379 тис. взаємодій, 1706 елементів та відсутністю інформації про користувачів. Головною перевагою реалізації є простота зберігання даних, та математичне підґрунтя.

## 1.3. Аналіз існуючих програмних рішень надання рекомендацій

### *1.3.1. Google Recommendations AI*

Recommendations AI дозволяє створювати високоякісні персоналізовані системи рекомендацій товарів, не вимагаючи високого рівня знань у галузі машинного навчання та проєктування систем. Recommendations AI надає можливості реалізації двох задач:

- Введення даних – ви можете завантажувати та керувати інформацією про каталог продуктів та журналами події користувачів веб-сайтів. Платформа використовує цю інформацію для навчання та оновлення моделей рекомендацій.
- Прогнозування – можливість запитувати рекомендації на основі каталогу товарів та журналів користувачів.

Переваги:

- швидке налаштування;
- конфігурація щодо надання рекомендацій;
- точність рекомендацій;
- інтеграція з іншими Google сервісами.

Недоліки:

- комплексність документації;
- відсутня інтеграція з більшістю мов програмування;
- відсутність авторизації через соціальні мережі.

### *1.3.2. QuarticON*

QuarticON – платформа що використовує AI, який оптимізує пропозиції електронної комерції, передбачуючи, які продукти більш імовірно придбають їх клієнти. Впровадження QuarticON в електронному магазині має головну перевагу – він забезпечує максимальну ефективність при мінімальних зусиллях при інтеграції. Створення персоналізованого досвіду покупок допомагає зменшити показник відмов, а також значно сприяє доходу. Також QuarticON надає доступ до відкритих модулів

написаних популярними мовами програмування, що полегшує інтеграцію системи в існуючі додатки.

Переваги:

- детальна документація;
- інтеграція з популярними мовами програмування;
- дані з соціальних мереж;
- простота інтеграції.

Недоліки:

- неможливість додавати свої конфігурації щодо надання рекомендацій;
- відсутність авторизації через соціальні мережі.

## 1.4. Результати проведення аналізу

Дані, отримані при детальному аналізі схожих за функціональними особливостями рішень, їх переваг та недоліків, свідчать про те, що жодне з розглянутих рішень в повній мірі не реалізує усіх потреб.

При цьому, кожне з існуючих рішень якісно реалізує якусь частину бажаних функціональних особливостей. Наприклад, Google Recommendations AI має чудову інтеграцію з Google сервісами, але через відсутність інтеграції з популярними мовами програмування є складною в використанні в нових проєктах. Але, QuarticON пропонує підтримку популярних мов програмування, але не надає інструменти налаштування рекомендацій. Також жодний з продуктів не використовує соціальні зв’язки між клієнта для побудови рекомендацій.

В результаті аналізу наявних аналогів, їх особливостей, переваг та недоліків було сформульовано список бажаних функціональних особливостей розроблюваної системи:

- система має надавати можливість реєстрації нового користувача;
- система має надавати можливість авторизації існуючого користувача;

- система має надавати можливість завантажувати товари;
- система має надавати користувацький інтерфейс у вигляді кабінета веб-сторінки;
- система має надавати виборку товарів, що побудована на основі соціальних зв’язків клієнтів;
- система має надавати HTTP API;
- система має надавати ключі доступу до HTTP API;
- система має надавати можливість перегляду останніх дій клієнтів;
- система має надвати можливість перегляду популярних товарів;
- система має надавати рекомендації на основі соціальних зв’язків;
- система має надавати відкриті модулі для інтеграції з мовами програмування;
- система має надавати можливості конфігурації алгоритма надання рекомендацій.

# 2. ОБГРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ

## 2.1. Вибір мови програмування

### *2.1.1. JavaScript*

JavaScript – це динамічна, нетипізована та інтерпретована мова програмування високого рівня. Була стандартизована згідно специфікації ECMAScript. Мова є високорівневою, отже не вимагає від розробника знання про те як працює зсередини програма нею написана. Також автоматично контролює пам’ять, тобто постачається з збиральником сміття. Є динамічною мовою програмування, отже дозволяє додавати нові абстракції під час виконання, що надає можливість писати гнучкий і універсальний код. Код написаний мовою JavaScript також називають скриптами.

На сьогодні скрипти написані даною мовою можуть бути запущені на будь-якому комп’ютері, що забезпечений двигуном JavaScript, тобто мова є кросплатформеною, отже може бути використана не тільки в браузері, а також в якості серверної мови програмування. Популярні двигуни:

- V8 – двигун розроблений компанією Google, підтримує популярні ОС, такі як Linux, MacOS, та використовується в браузерах Chrome та Opera;
- SpiderMonkey – двигун, що використовується в браузері Firefox.

### *2.1.2. TypeScript*

TypeScript – це об'єктно-орієнтована мова з відкритим кодом, розроблена та підтримувана корпорацією Майкрософт, ліцензована за ліцензією Apache 2. Це є налаштування над популярною мовою програмування JavaScript, вони є зворотно-сумісні.

За останні кілька років у JavaScript спостерігається величезний розвиток. Це найбільш універсальна міжплатформна мова, яка використовується для розробки сучасних додатків. Він може використовуватися для розробки як клієнтської програми додатку, з такими

фреймворками, як Angular або ReactJS, так і з боку сервера, з такими платформами, як NodeJS. Зараз мова вважається найбільш обраною мовою для побудови прикладних програм. Але JavaScript ніколи не був призначений для розробки масштабних додатків. JavaScript – це динамічна мова програмування без системи типів, на відміну від добре структурованих та вдосконалених мов, таких як C# або Java. Система без типів означає, що змінна в JavaScript може мати будь-який тип значення, такий як рядок, число, булеве значення і інші. Важливо що помилки можна виявити під час компіляції, а не під час виконання.

Без системи типів важко масштабувати JavaScript для створення складних додатків з великими командами, що працюють над одним кодом.

Отже, причиною використання TypeScript є те, що він дає можливість використовувати JavaScript в масштабі.

TypeScript компілюється в простий JavaScript. Компілятор TypeScript також реалізований у TypeScript і може використовуватися з будь-яким браузером або JavaScript двигунами на зразок NodeJS. TypeScript для компіляції потребує сумісне середовище ECMAScript 3 або вище. Це умова, яку сьогодні задовольняють усі браузери. Деякі найпопулярніші фреймворки JavaScript, такі як Angular.js і WinJS, написані на TypeScript.

### *2.1.3. Ruby*

Ruby – динамічна типізована мова програмування з відкритим кодом з акцентом на простоту та продуктивність. Мова є повністю об’єктно-орієнтованою – відсутні примітивні типи даних.

Ruby найчастіше використовується для створення веб-додатків. Однак це мова загального призначення, схожа на JavaScript, тому вона має багато інших застосувань, таких як прототипування, аналіз даних, написання системних утиліт. Зазвичай використовується в поєднані з фреймворком Ruby on Rails, який розроблений для Ruby. Homebrew, надзвичайно популярний інструмент для встановлення програмних пакетів на macOS, також написаний на Ruby. Також як і програмне забезпечення для

безпеки Metasploit, яке дозволяє протестувати захист веб-сайті та програм від хакерських атак.

Переваги мови програмування Ruby:

- велика кількість синтаксичний структур, що роблять вихідний код простим в розумінні;
- фреймворк Ruby on Rails, що побудований розробниками мови, включає велику кількість абстракцій для побудови веб-додатків.

## 2.2. Вибір фреймворку для розроблення серверної частини проєкту

### *2.2.1. NodeJS*

NodeJS – це платформа, побудована на JavaScript, що використовується браузером Chrome для створення швидких та масштабованих мережевих додатків. NodeJS використовує керовану подіями модель, що не блокує введення / виведення. Це робить її легкою та ефективною, ідеальною для використання в real-time додатках, які працюють на розподілених пристроях. Також це проєкт з відкритим кодом, міжплатформне середовище виконання JavaScript програм, які можна виконувати на більшості операційних систем: OS X, Microsoft Windows та Linux.

NodeJS також надає багату бібліотеку різноманітних модулів JavaScript, що значно спрощує розробку додатків, що використовують NodeJS.

Переваги:

- Асинхронна та керована подіями – всі API бібліотеки NodeJS асинхронні, тобто не блокують потік виконання. Це означає, що додаток на базі NodeJS ніколи не чекає повернення даних з API. Після одного виклику до API він переходить до наступного, а механізм сповіщень допомагає додатку отримати відповідь від попереднього виклику API.

- Швидкість – NodeJS побудована на Google V8 JavaScript Engine, тому бібліотека швидка у виконанні коду.
- Однопоточність, але при цьому масштабованість – NodeJS побудована на моделі з одним потоком виконання, але з механізмом циклу подій. Події допомагають додатку реагувати не блокуючи способом і робить додаток високо масштабованим на відміну від традиційних додатків, які створюють обмежені потоки для обробки запитів. Не зважаючи на те, що NodeJS є одно поточною, вона може обробляти більшу кількість запитів, ніж традиційні сервери, такі як Apache HTTP Server.
- Відсутність буферизації – NodeJS ніколи не створює буфер даних, а оброблює дані окремими шматками.

### *2.2.2. NestJS*

Nest – це фреймворк для створення масштабованих, простих в тестуванні, підтримуваних та ефективних додатків на сервері NodeJS. NestJS використовує неймовірну популярний та надійний JavaScript, і був створений за прикладом таких фреймворків, як Angular, React та Vue. Він повністю підтримує TypeScript за замовчанням.

Реалізація Nest використовує фреймворки HTTP-серверів, такі як Express (за замовчуванням) та Fastify. Nest забезпечує рівень абстракції над цими бібліотеками, але також може надавати свої API безпосередньо розробнику. Це дозволяє легко використовувати безліч сторонніх модулів, які доступні для кожної платформи.

Переваги Nest над іншими фреймворками:

- підтримка TypeScript;
- Nest CLI для ініціалізації та розробки ваших програм;
- відмінна документація;
- модульне тестування та інтеграційні тести;
- масштабується завдяки універсальності абстракцій;

- створений для великих масштабів корпоративних програм (nx.dev використовує NestJS);
- відкритий код (ліцензія MIT);
- підтримка мікросервісної архітектури.

## 2.3. Вибір бібліотек для розроблення клієнтської частини проєкту

### *2.3.1. ReactJS*

ReactJS – це бібліотека JavaScript, яка поєднує в собі швидкість JavaScript та використовує новий спосіб візуалізації веб-сторінок, роблячи їх динамічними та чутливими до вводу користувачів. Продукт значно змінив підхід Facebook до розробки. Після випуску бібліотеки як інструмент JavaScript з відкритим кодом у 2013 році вона стала надзвичайно популярною завдяки революційному підходу до програмування користувальницьких інтерфейсів.

Команді ReactJS вдалося збільшити швидкість оновлень сторінки за допомогою віртуального DOM дерева. На відміну від інших фреймворків, які працюють з реальним деревом, ReactJS використовує свою абстрактну копію. Він оновлює навіть мінімалістичні зміни, застосовані користувачем, але не впливає на інші частини інтерфейсу. Це також можливо завдяки ізоляції компонентів ReactJS.

Переваги ReactJS над іншими JavaScript клієнтськими бібліотеками:

- швидкість, реалізація віртуальної DOM дерева та різні оптимізації рендеру та оновлення сторінки;
- підтримка серверного рендерингу;
- React реалізує концепції функціонального програмування, що робить компоненти легкими для тестування і пере використання;
- проста міграція між версіями, розробники бібліотеки надають інструменти для швидкого переходу між релізами.

### *2.3.2. Redux*

Redux – це контейнер стану для додатків JavaScript. Він допомагає писати програми, які ведуть себе послідовно, запускаються в різних середовищах (клієнт, сервер та рідне місце), і їх легко протестувати.

Бібліотека була створена на основі функціонального стилю програмування. Отже в бібліотеці є лише кілька простих абстракцій, які являються простими абстракціями мови JavaScript:

- стан – об’єкт;
- дія (провокує зміну стану) – об’єкт;
- обробник дій (змінює стан) – функція.

Отже, код написаний з використанням цієї бібліотеки є простим в тестуванні та масштабуванні, а його компоненти можна повторно використовувати.

## 2.4. Вибір системи керування базами даних

### *2.4.1. MongoDB*

MongoDB – це база даних, орієнтована на документи. Дані організовані у вигляді документів JSON (еквівалент рядків) із полями (стовпці еквівалентні), які згруповані у колекції (еквівалентні таблиці). Вона використовує формат BSON для зберігання документів (бінарний серіалізований JSON), який розширює реалізацію JSON, пропонуючи додаткові типи даних (наприклад, бінарний). Вона також забезпечує перевірку даних на основі стандарту схеми JSON (при налаштуванні колекції ви можете надати визначення схеми JSON).

MongoDB не має схеми за дизайном, тобто кожний документ може мати свій набір унікальних полів у межах однієї колекції. Крім того, вона легко масштабується вертикально та горизонтально задля кращої оптимізації.

База даних, орієнтована на документи, має певні переваги: гнучкість (відсутність жорсткої структури), добре підходить для сучасних фреймворків JavaScript (безпосереднє використання JSON), велика обробка даних та статистика / аналіз даних у режимі реального часу. Реляційні бази даних, з іншого боку, забезпечують суворе дотримання цілісності даних та надійний спосіб поєднання записів під час отримання.

Отже, MongoDB буде використовуватись для зберігання незв'язаних даних.

### *2.4.2. Neo4j*

Neo4j – це система управління базами даних, що зберігає дані в виді графів. Підтримує ACID транзакції. Кожний вузол та зв'язок може зберігати в собі дані. Поставляється разом з мовою запитів Cypher, що розроблена для роботи з графами. Також підтримує режим роботи Master-Slave. Переваги Neo4j:

- висока продуктивність при великих об'ємах зв'язаних даних;
- постачається з клієнтським інтерфейсом для тестування запитів;
- аналіз навантаження та оптимізація вузлів в реальному часі.

## 2.5. Хостинг сервіс

В якості хостинг сервісу обрано Heroku, що використовується для розміщення та запуску додатків та фонових процесів. Heroku використовують в основному розробники програмного забезпечення, так як він є легкий в налаштуванні. Також надає можливість використовувати плагіни – окремі додатки, що включають в себе готові конфігурації деяких часто використовуваного програмного забезпечення, наприклад СУБД, проксі сервера та інше. Всі розміщенні сервіси на даному хостингу по замовчанню отримують системи моніторингу та журналювання. Команда Heroku також розроблює продукт Heroku CLI, який є безкоштовним та підтримується більшістю операційних систем, застосовується для віддаленого керування, розгортання та перегляду результатів моніторингу.

Даний хостинг є найкращим варіантом для проєкту, через його простоту, швидкість налаштування та можливості безпроблемного масштабування.

## 2.6. Середовище розробки

В якості середовища розробки було обрано WebStorm, що комплексне середовище розробки для сучасної розробки з JavaScript: клієнтська, серверна та мобільна. Окрім переваг, які він надає розробникам, таких як економія часу на автоматизацію рутинних завдань, пошук та виправлення помилок, використання інтелектуальної підтримки IDE та підвищення загальної продуктивності.

Переваги:

- підтримує розповсюджені фреймворки та бібліотеки;
- також вирішує вузько направлені та не розповсюджені проблеми;
- широко конфігурується;
- низькі системні потреби;
- плагіни – надають можливість додавати нову функціональність в середовище розробки;
- інтелектуальне авто доповнення коду;
- інтеграція з системним командним рядком;
- авто виправлення граматичних помилок в ідентифікаторах.

Дане середовище є фаворитом через його гнучкість в налаштуванні, також він підтримує фреймворки та бібліотеки, що використовуються при розробленні проєкту.

## 2.7. Додаткові технології та бібліотеки

### *2.7.1. Eslint*

ESLint – це утиліта для перевірки якості коду JavaScript з відкритим кодом, створена Nicholas C. Zakas у червні 2013 року. Лінтування – це тип статичного аналізу, який часто використовується для пошуку проблемних

шаблонів коду, який не дотримується певних стильових правил. Для більшості мов програмування існують інструменти для лінтування коду, а компілятори цих мов мають окремий етап виконання для запуску перевірки. JavaScript, будучи динамічною та слабко типізованою мовою, особливо схильна до помилок. Через те що JavaScript не компілюється, єдиний спосіб перевірки коду це статичний аналіз або лінтування. Такі інструменти, як ESLint, дозволяють розробникам виявляти проблеми з кодом, не виконуючи його.

Основна причина створення ESLint була в тому, щоб дозволити розробникам створювати власні правила перевірки. ESLint розроблений так, щоб усі правила перевірки були повністю модульними. Правила за замовчуванням також є модульними, по аналогії з користувацькими правилами. Всі вони можуть слідувати одній і тій же схемі, як для самих правил, так і для тестів. У той час як ESLint постачає деякі вбудовані правила, для того щоб зробити його зручним, можна доповнювати правила по ходу виконання проєкту.

### *2.7.2. Prettier*

Prettier – інструмент для форматування коду. На сьогодні головна причина вибору Prettier – це шлях до припинення всіх дебатів щодо правил форматування коду. Через те що екосистема JavaScript є неоднорідна, і не має єдиного підходу та стандартів форматування коду, постає питання того як повинен форматуватись код на проєкті. Отже, наявність однорідного стилю коду є важливий аспект для проєкту та команди. Інструмент постачається з стандартними налаштуваннями стилю з можливістю кастомізації. Prettier підтримують всі найпопулярніші IDE та текстові редактори, такі як:

- Webstorm;
- Atom;
- Visual Studio Code.

Також інструмент можна налаштувати для автоматичного форматування коду.

### *2.7.3. Lerna*

Lerna – це інструмент, який оптимізує робочий процес управління пакетами.

Розбиття великих баз кодів на окремі незалежно розроблені пакети є надзвичайно корисним для спільного використання коду. Однак внесення змін відразу у декількох середовищах зберігання пакетів є комплексним процесом, який включає в себе велику кількість монотонної роботи, кожний окремий пакет потрібно налаштовувати та актуалізувати окремо.

Для вирішення цих (та багатьох інших) проблем деякі проєкти організовують свої кодові бази в багато пакетні репозиторії. Такі проєкти, як Babel, React, Angular, Ember, Meteor, Jest та багато інших, розробляють усі свої пакети в одному репозиторії.

## 2.8. Висновки до розділу

У даному розділі розглянуто основні технології та інструменти для створення веб-застосунків.

Основною мовою програмування було обрано JavaScript для клієнтської частини системи та налаштування над цією мовою TypeScript для серверної. JavaScript є нетипізованою мовою, отже її доцільно використовувати при побудові простих інтерфейсів, також більшість клієнтських бібліотек можуть постачатись без підтримки типів TypeScript. В якості клієнтського фреймворка було обрано ReactJS що буде використовуватись разом з мовою програмування JavaScript. Серверним фреймворком було обрано NestJS через те що він написаний мовою TypeScript, та вирішує більшість архітектурних проблем. Середовище розробки WebStorm підтримує обрані клієнтські та серверні фреймворки. Хостинг сервісом було обрано Heroku через простоту налаштування та підтримку обраних мов програмування.

# 3. ОПИС РОЗРОБЛЕНОЇ СИСТЕМИ

## 3.1. Загальний опис

Дана система є веб-сервісом, що має набір функціональних можливостей та відповідає сучасним стандартам користувацьких інтерфейсів та інтерфейсів для сторонніх розробників.

В системі визначено три діючих ролі:

- користувач – адміністратор, що бажає інтегрувати систему рекомендацій в власний додаток, та переглядає статистику;
- користувач-розробник – підвид користувача, що працює з HTTP API веб-сервіса задля інтеграції системи рекомендацій;
- клієнт – користувач соціальної мережі, що переадресований з додатку користувача.

Задля отримання послуг сервісу, користувач може використовувати два інтерфейсу, один задля перегляду рекомендацій та популярних товарів, інший для інтеграції користувацьких додатків:

- HTTP API – створення дій клієнтів, авторизація клієнтів, отримання рекомендацій (персоналізованих виборок товарів) та створення товарів;
- браузерний клієнтський інтерфейс – перегляд останніх дій клієнтів, популярних товарів та отримання ключів доступу до HTTP API.

Користувач повинен обов’язково авторизуватись задля користування браузерним інтерфейсом, також він повинен отримати ключі доступу до HTTP API веб-сервісу, та надавати їх в кожному запиті задля ідентифікації.

Задля отримання персоналізованої клієнтської рекомендації користувач-розробник повинен виконати визначену послідовність дій:

1. Отримати ключі доступу до HTTP API.

2. Переадресувати клієнта за допомогою запита на HTTP API задля авторизації через соціальну мережу.
3. Отримати виборку товарів що рекомендуються конкретному користувачу за допомогою HTTP API.
4. Надсилати інформацію про дії користувача задля отримання більш точних рекомендацій.

Особливою можливістю для кожного користувача додатку є створення власних товарів, дій клієнта та зміна пріоритетності між діями клієнта задля надання майбутніх рекомендацій. Використовуючи HTTP API користувач може:

- створювати товар;
- створювати дію клієнта;
- отримувати персоналізовану рекомендацію для окремого клієнту;
- авторизувати клієнта через соціальні мережі;
- змінити пріоритет критерію оцінювання.

Браузерний клієнтській інтерфейс надає статистичні можливості та налаштування доступу, такі як:

- перегляд найпопулярніших товарів;
- отримання ключів доступу до HTTP API;
- перегляд існуючих товарів;
- перегляд існуючих критеріїв оцінювання.

## 3.2. Опис вимог до розроблюваної системи

Чітке визначення вимог є важливим для успіху проєкту. Вони встановлюють офіційну угоду між клієнтом і виконавцем про те, що вони обидва працюють для досягнення однієї мети. Якісні, детальні вимоги також допомагають пом’якшити фінансові ризики та тримати проєкт за графіком.

Створення вимог є складним завданням, оскільки воно включає набір процесів, таких як випробовування, аналіз, специфікація, перевірка та управління.

Перш ніж визначати вимоги до проєкту, сформулюємо типи можливих вимог. Класифікацію вимог наведено на рис.1.

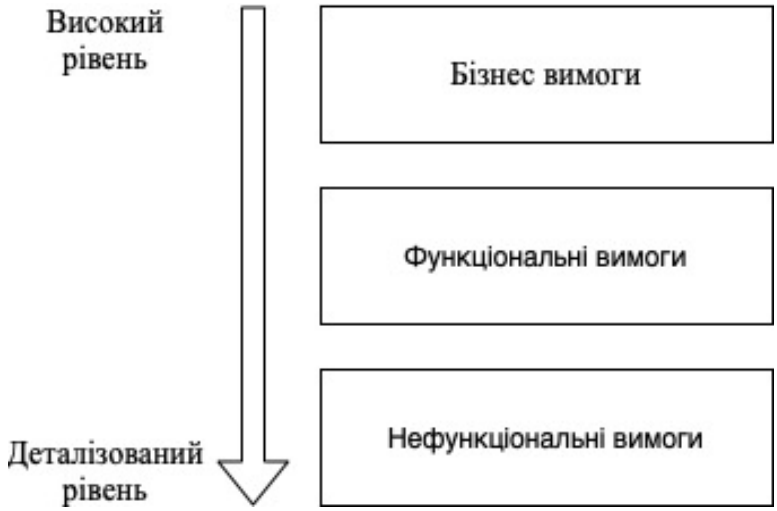



Рис. 1. Класифікація вимог

Можна виділити 3 основних види вимог:

- Бізнес вимоги – до них належать твердження на високому рівні про цілі, завдання та потреби.
- Нефункціональні вимоги – описують загальні характеристики системи. Вони також відомі як атрибути якості.
- Функціональні вимоги – описують, як повинен поводитись продукт, які його особливості та функції.

Функціональні вимоги – це функції продукту, які розробники повинні реалізувати, щоб користувачі могли виконати свої завдання. Отже, важливо зробити їх зрозумілими як для команди розробників, так і для зацікавлених сторін. Як правило, функціональні вимоги описують поведінку системи в конкретних умовах.

Нефункціональні вимоги зв’язані із такими системними характеристиками як час відгуку та надійність. Вони з’являються через

наявність певних корпоративних політик, користувацьких вимог, бюджетних обмежень, тощо і не пов’язані прямо із якоюсь конкретною функцією системи. Розроблюване ПЗ має відповідати встановленим нефункціональним вимогам для того, аби зробити його роботу більш ефективною.

В результаті проведення аналізу предметної галузі було сформульовано такий набір функціональних вимог до розроблюваного ПЗ:

1. Система має надавати можливість зареєструвати нового користувача та авторизувати існуючого користувача.
2. Система має надавати можливість авторизувати клієнта через соціальну мережу.
3. Система має надавати ключі доступу до HTTP API користувачам-розробникам.
4. Система має надавати можливість переглянути з можливістю фільтрації та сортування:
    - списку товарів;
    - списку критеріїв оцінювання та їх пріоритети;
    - останні дії клієнтів.
5. Система має надавати можливість користувачам-розробникам за допомогою HTTP API:
    - додавати новий товар;
    - створити нову дію та пов’язати її с товаром;
    - розставити пріоритети між критеріями оцінювання (дії користувача);
    - отримувати рекомендовані товари під конкретного клієнта;
    - реєструвати нового клієнта через перенаправляючий запит на соціальну мережу.

Опишемо нефункціональні вимоги до розроблюваного ПЗ. Вимоги до продуктивності описані в табл. 3.1, вимоги до реалізації до безпеки – в табл. 3.2, вимоги до реалізації – в табл. 3.3.

Таблиця 3.1

Вимоги до продуктивності

| Код | Вимога |
|---|---|
| P-1 | Знаходження рекомендацій для клієнта не повинно продовжуватись більше ніж 400 мілісекунд. |
| P-2 | Система повинна масштабуватись вертикально та горизонтально |
| P-3 | Система повинна оброблювати 500 запитів одночасно. |
| P-4 | Вихідний код клієнтської частини повинен пройти перетворення мінімізації. |

Таблиця 3.2

Вимоги до безпеки

| Код | Вимога |
|---|---|
| S-1 | Паролі користувачів повинні бути захищені від зчитування, отже захешовані. |
| S-2 | Користувач повинен бути захищений від браузерних атак (CORS attack, Clickjacking). |
| S-3 | Система повинна перевіряти всі вхідні данні, що надходять від користувача. |
| S-4 | Система повинна бути захищена від серверних атак (XSS, SQL Injection, CSRF). |
| S-5 | Вихідний код клієнтської частини повинен пройти перетворення обфускації |

Таблиця 3.3

Вимоги до реалізації

| Код | Вимога |
|---|---|
| I-1 | Клієнтський додаток повинен підтримуватись поширеними браузерами та їх версіями до 2015 року включно. |

Продовження табл. 3.3

| | |
|---|---|
| І-2 | Код клієнтського додатку повинен бути мінімізований та не займати більше ніж 3 мегабайта браузерного місця. |
| І-3 | Серверна частина додатку має бути розгорнута на віддаленому сервері з ОС Ubuntu Linux |

### 3.3. Архітектура програмного забезпечення

Для взаємодії браузерного веб-клієнту та сторонніх додатків користувачів була обрана клієнт-серверна архітектура. Клієнт-сервер – архітектура комп'ютерної мережі, в якій багато клієнтів (віддалені комп'ютери) запитують та отримують послугу від централізованого сервера (хост-комп'ютера).

Клієнтські комп'ютери надають інтерфейс, що дозволяє користувачеві комп'ютера запитувати послуги сервера та відображати результати, які сервер повертає. Сервери чекають надходження запитів від клієнтів, а потім відповідають на них. В ідеалі, сервер забезпечує стандартизований прозорий інтерфейс для клієнтів (в нашому випадку це HTTP API) щоб клієнти були інкапсульовані від специфіки системи (тобто апаратного та програмного забезпечення), що надає послугу.

Клієнти часто знаходяться на робочих станціях або на персональних комп'ютерах, а сервери розташовані в інших місцях мережі, як правило, на більш потужних машинах. Ця обчислювальна модель особливо ефективна, коли клієнти та сервер мають різні завдання, які вони регулярно виконують.

На проєкті клієнтами виступають додатки клієнтів та браузерна клієнтська частина проєкту, NodeJS додаток в якості сервера. Архітектура обрана через її широку розповсюдженість та простоту реалізації, також за відсутністю аналогічних архітектур, які могли бути використані для побудови веб-додатку. Взаємодію серверу с клієнтом наведено на рис. 2.

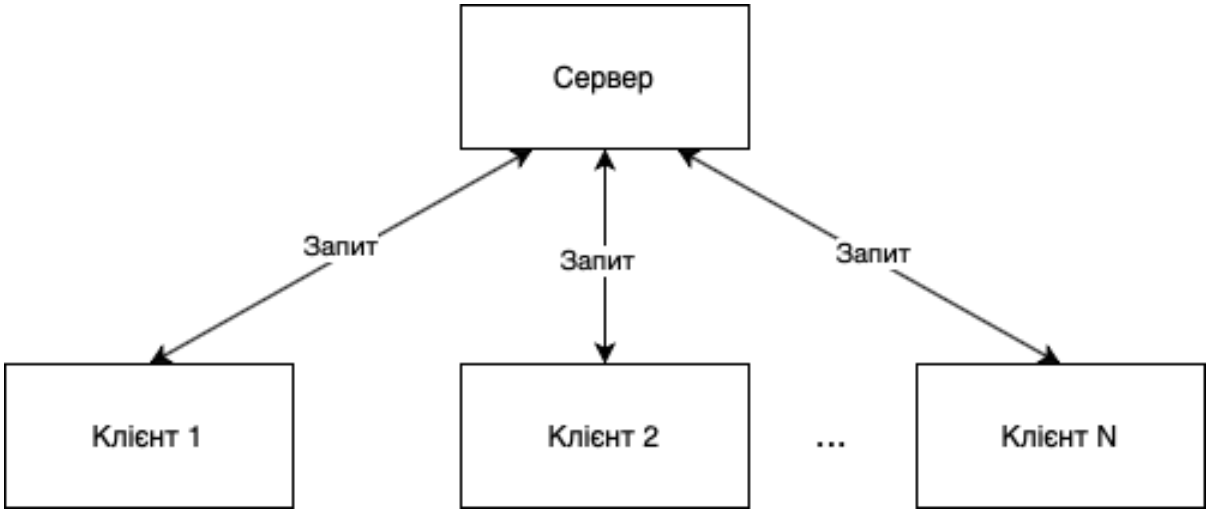



Рис. 2. Клієнт-серверна взаємодія

Через те, що обраний серверний фреймворк NodeJS є однопоточним, проблема масштабування лягає на архітектуру. Одним із популярних та практичних підходів масштабування є декомпозиція додатку на мікросервіси.

Мікросервіси – це підхід до архітектури програмного забезпечення, в якому додаток складається з безлічі дрібних компонентів, кожен з яких виконує одну функцію, таку як автентифікація, повідомлення або обробка платежів. Кожний мікросервіс є окремою незалежною частиною програмного забезпечення, має власну кодову базу, інфраструктуру та, інколи, базу даних. Мікросервіси працюють разом, взаємодіючі через веб-API або черги для обміну повідомленнями, щоб реагувати на вхідні події.

Мікросервіси перетворюють комплексний монолітний додаток на набір сервісів, які швидше розробляються і простіше підтримуються. Кожен з цих сервісів також може бути розроблений незалежною командою, яка орієнтована на окрему послугу. Це також дозволяє безперервно масштабуватись, оскільки кожен мікросервіс можна розгорнути незалежно.

Тому мікросервіси є:

- прості в експлуатації та підтримці;
- не пов’язані один з одним;

- незалежні один від одного, тому розгортаються незалежно;
- побудовані за функціональними потребами, наприклад, сервіс авторизації, сервіс доставки пошти.

Отже після дослідження методів організації системних компонентів, була спроєктована наступна архітектура, подана на рис. 3.

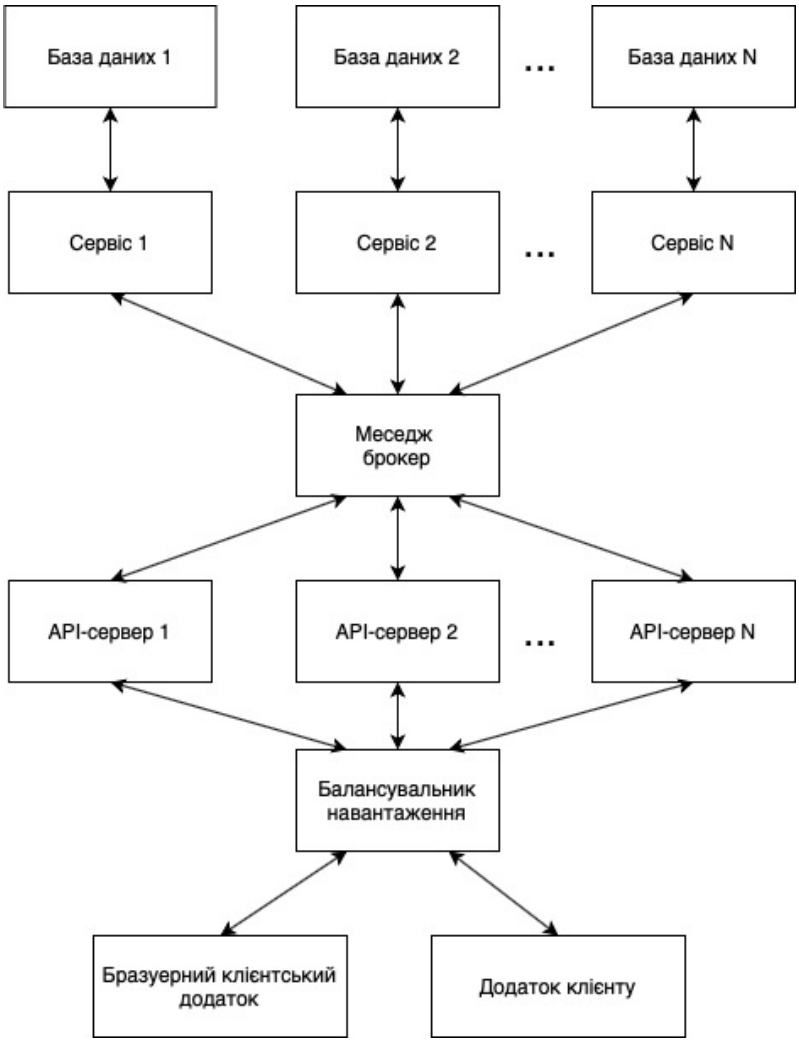



Рис. 3. Архітектура розроблюваної системи

Розглянемо архітектуру детальніше, вона складається з 6 рівнів (від клієнта до даних):

- клієнти – в якості клієнтів можуть виступати додатки клієнтів, що мають до доступ до рекомендацій, також клієнтський веб-додаток;

- балансувальник навантаження – находиться між API-серверами і клієнтами та балансує навантаження на кожний з серверів, вирішуючи який сервер буде відповідати на окремий запит клієнта;
- API-сервера – місце знаходження бізнес логіки, міст між клієнтами та послугами, наприклад, авторизація, отримання рекомендації;
- меседж брокер – доставляє повідомлення між сервісами, та відповідає за взаємодію API-сервера з сервісами;
- сервіси – кожен сервіс відповідає за окрему послугу (див. вище);
- бази даних – під кожний кластер сервісів є окрема база даних, можливість виконання одночасних запитів до бази забезпечена транзакційністю запитів, тобто будь-який запит (або послідовність запитів) сприймається базою даних як єдина та не розривна одиниця роботи.

В системі розроблено 4 мікросервіси: API, Users, Recommendations, та Auth, кожен з яких має доступ до своєї бази даних. Наявні мікросервіси системи наведені на рис. 4.

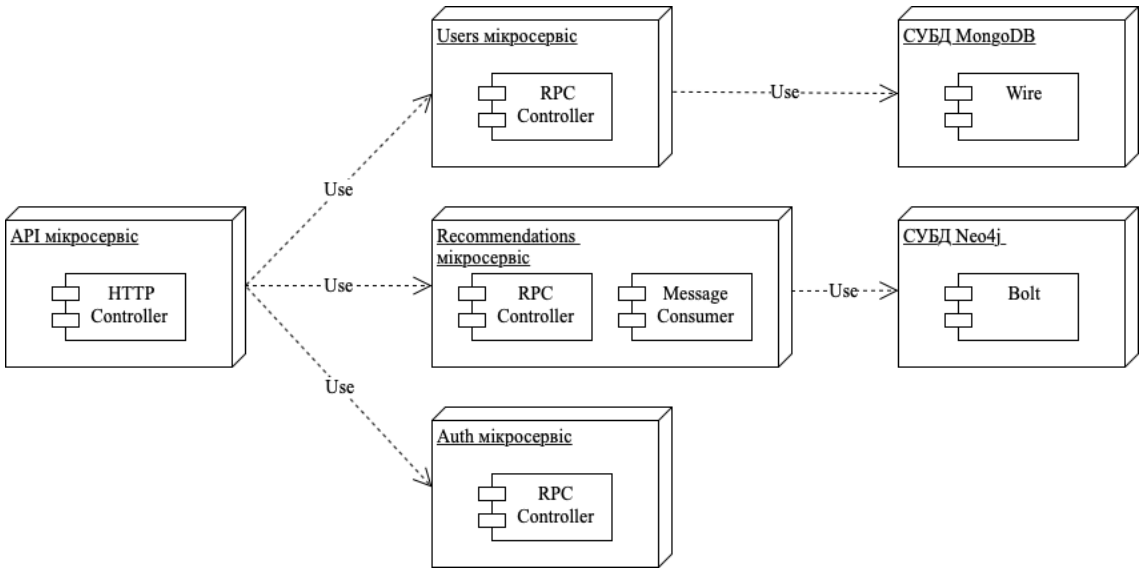



Рис. 4. Наявні мікросервіси

### 3.4. Алгоритм надання рекомендації

Перш ніж система зможе надавати клієнту користувача рекомендації, клієнт повинен авторизуватись в соціальній мережі та надати доступ до даних, що з ним пов’язані (друзі, контакти).

Після отримання доступу до зв’язків клієнта всередині соціальної мережі, система збереже данні клієнта та спробує знайти вже існуючих клієнтів в рамках системи. В результаті кількох авторизацій та за допомогою отриманих даних буде побудовано граф зв’язаних клієнтів що можуть бути пов’язані один з одним. Також в результаті авторизації клієнт повинен бути авторизований на стороні додатку користувача.

Разом з сформованими зв’язками будуть збережені такі дані про клієнтів, які можуть бути також використані для надання рекомендацій, та будуть виступати критеріями оцінювання:

- вік клієнта;
- місце проживання клієнта;
- гендерна приналежність клієнта.

При кожній дії клієнта (покупка товару, вподобання), користувач-розробник повинен зробити запит на HTTP API системи з вказаним ідентифікатором клієнта та дією яку він виконав. На основі дій та зв’язків між клієнтами буде побудовано граф, який буде використано для надання подальших рекомендацій. Приклад такого графу наведено на рис. 5.

Отже, після побудови графу клієнтів, їх зв’язків один з одним та товарами, користувач-розробник зможе зроби запит на HTTP API системи задля отримання рекомендації що буде включати такі дані:

- критерій або критерії за якою буде надана рекомендація, наприклад “за покупкою”;
- за якою дією, наприклад “покупка”;
- дані для фільтрації товарів (залежить від наданих товарів).

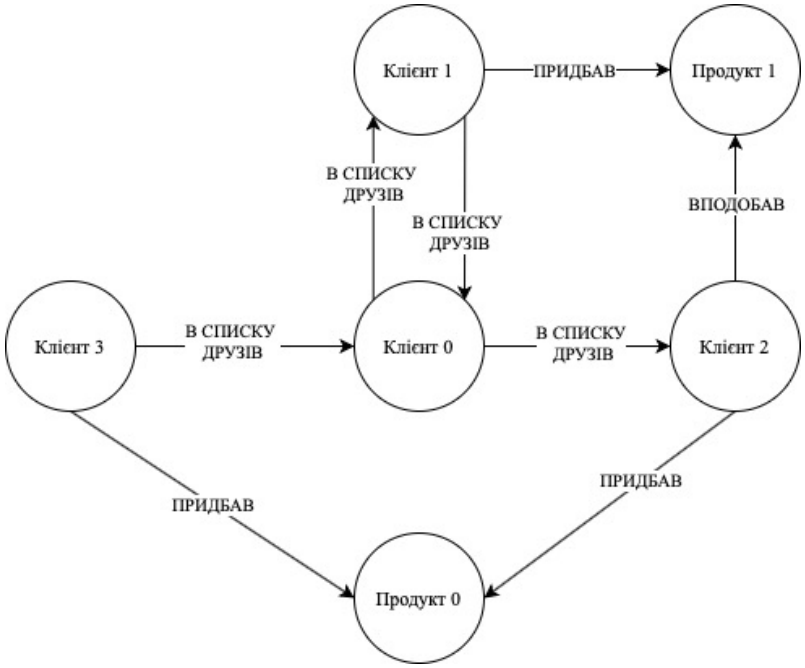



Рис. 5. Приклад побудованого графу

Запит на пошук рекомендацій буде виконаний за допомогою мовою Cypher, що використовується задля пошуку та отримання даних з графових СКБД.

Лістинг 3.1 Код запиту отримання рекомендації

```
MATCH (c:Client {socialNetworkID: $socialNetworkID})-[f:FRIEND * 1..]->(n)-
[:PURCHASED]->(p: Product)
RETURN p, length(f) AS depth
ORDER BY depth
```

## 3.5. Опис структур даних системи

В результаті проведеного аналізу сформульованих вимог вирішено, що дані будуть зберігатись в двох базах даних:

- ненормалізована СУБД (MongoDB);
- графова СУБД (Neo4j).

Структури даних ненормалізованої бази даних:

1. Користувач (User) складається з таких полів:
    - Ідентифікатор ( _id);

- Пароль (password);
- Електронна пошта (email);
- Ключ до HTTP API (apiKey);
- Дата створення (createdAt).

2. Пріоритет дії клієнта (ActionPriority) складається з наступних полів:
   - Дія (actionName);
   - Пріоритет (priority);
   - Користувач (user) – ідентифікатор користувача.
3. Компанія (Company) складається з таких полів:
   - Користувач (user) – ідентифікатор користувача;
   - Посилання (url) – посилання на сайт;
   - Посилання для авторизації (authUrl).

Структури даних графової бази даних:

1. Вузли:
   - Клієнт (Client) містить такі поля:
     - Ідентифікатор соціальної мережі (socialNetwork);
     - Ідентифікатор в соціальній мережі (socialNetworkId);
     - Ідентифікатор (id);
     - Вік (age);
     - Гендер (gender).
   - Товар (Product) містить наступні поля:
     - Тип (kind);
     - Ідентифікатор (id);
     - Ідентифікатор компанії (companyId).
2. Відношення:
   - Дія (Action) – відношення між клієнтом та товаром:
     - Назва (kind);
     - Дата створення (createdAt).

- Контакт (Related) – відношення між двома пов’язаними клієнтами.

## 3.6. Висновки до розділу

В цьому розділі було представлено опис розробленої системи створення рекомендацій на основі соціальних зв’язків між клієнтами.

Були представлені функціональні та не функціональні вимоги до веб-сервісу. Всі вимоги були реалізовані у системі.

Було розглянуто архітектуру веб-сервісу, що є клієнт-серверною та мікросервісною, тому серверна частина додатку масштабується горизонтально та вертикально. Комунікація між мікросервісами відбувається за домогою повідомлень або віддаленого виклику процедур, що робить їх кросплатформенними та відмовостійкими.

Також було розглянуто структури даних системи, що зберігаються в графових та документарних базах даних, що є ефективним при використанні великої кількості пов’язаних даних.

# 4. АНАЛІЗ РОЗРОБЛЕНОЇ СИСТЕМИ

## 4.1. Особливості реалізації взаємодії між мікросервісами

Серверна частина реалізована у вигляді мікросервісного додатку, що складається з множини NodeJS додатків, які взаємодіють за допомогою повідомлень через меседж-брокер, або за допомогою виклику віддалених процедур.

Взаємодія повідомленнями відбувається за допомогою запису та зчитування повідомлень з черги, шо знаходиться на меседж-брокері, являється буфером повідомлень, який обмежений лише оперативної пам’яттю та розміром диска хоста. Обробка, відправка, зчитування повідомлень відбувається за протоколом AMQP (Advanced Message Queuing Protocol), за протокол меседж-брокер здійснює маршрутизацію повідомлень, гарантує доставку, розподіляє потоки даних, забезпечує можливість підписатись на потрібні типи повідомлень. Передусім повідомлення направляються в точку обміну, яка розподіляє їх по чергам, що до них прив’язані, при цьому вони там не зберігаються. Точки обміну можуть бути трьох типів:

- розгалуження – повідомлення передається в усі черги, що прив’язані до точки;
- прямі – повідомлення передається в чергу з ключем маршрутизації, що вказується при відправці;
- теми – повідомлення передається в чергу, в який зберігається маска, що збігається ключем маршрутизації.

Повідомлення зберігаються в чергах згідно з налаштування меседж-брокера, вони можуть видалятись з черги при успішному зчитуванні повідомлення клієнтом, або при закінченні терміну дії повідомлення. Схему оброблення повідомлення наведено на рис. 6.

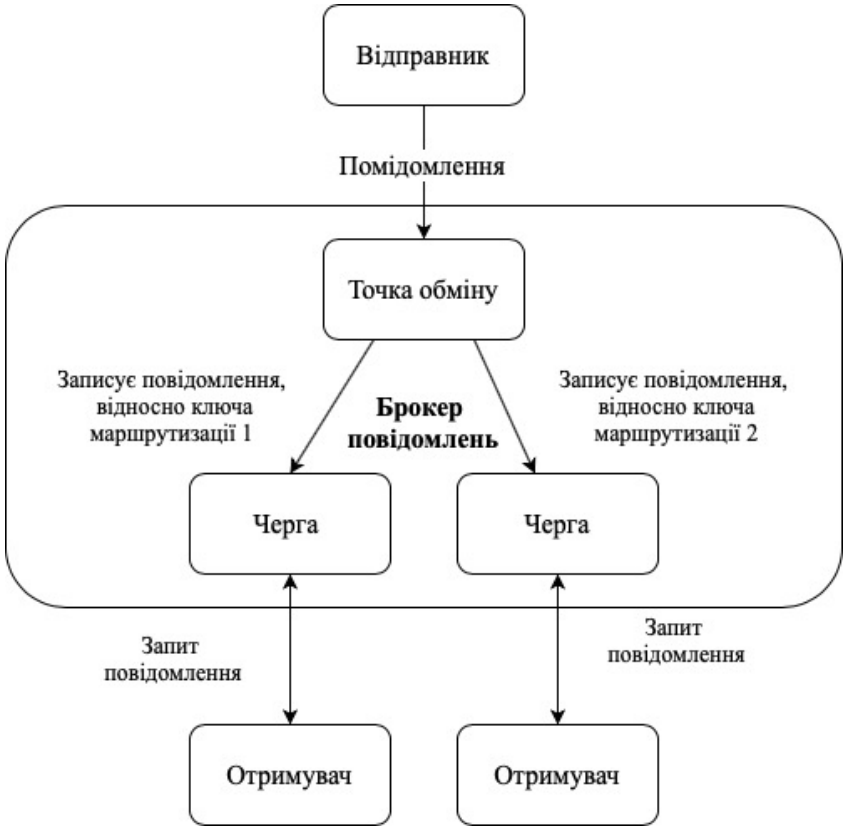



Рис. 6. Схема оброблення повідомлення

Виклик віддалених процедур – протокол що дозволяє програмі, яка запущена в одній мережі викликати процедуру програми в іншій мережі. Взаємодія за віддаленим викликом процедур відбувається за допомогою системи gRPC (Google Remote Procedure Calls). Система використовує протокол HTTP/2 для транспортування та Protocol Buffers в якості мови описування інтерфейсів, що дозволяє створювати платформонезалежні інтерфейси. Система надає можливість авторизації запитів, може розділяти та блокувати потоки даних. Схему виклику віддалених процедур наведено на рис. 7.

Головною перевагою виклику віддалених процедур є незалежність протоколу від платформи, та спорідненість такого підходу з парадигмою об’єктно-орієнтованого програмування, а саме наявність процедур.

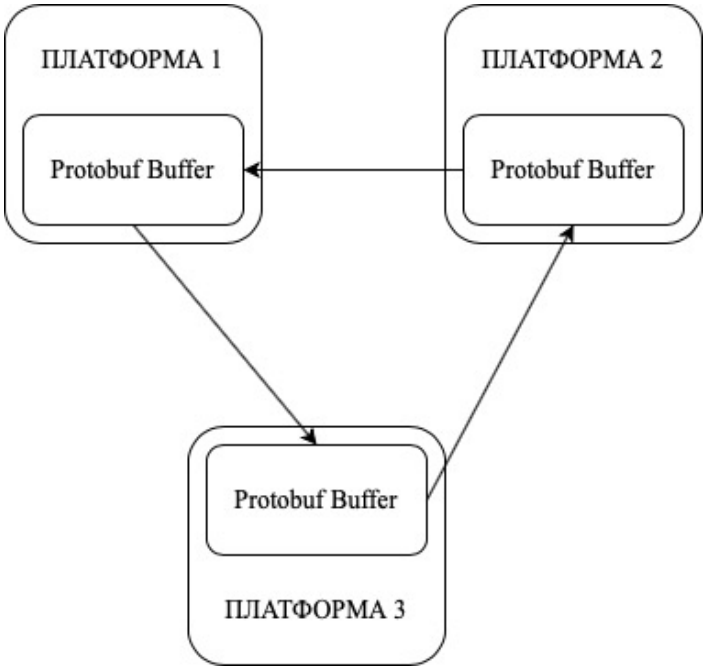



Рис. 7. Виклик віддалених процедур

## 4.2. Особливості реалізації авторизації користувача

Авторизація користувача в системі відбувається за допомогою стандарту JWT (JSON Web Token). Авторизація за допомогою JWT токену є клієнтською, тобто сервер не зберігає стан кожного клієнта в даний момент часу, отже токен зберігає всю потрібну інформацію задля авторизації користувача. Токен складається з трьох частин:

- заголовок – зберігає ім’я алгоритму за допомогою якого токен був закодований;
- метадані – клієнтські дані, які сервер додає до токену, можуть містити будь-яку інформацію;
- підпис – заголовок та метадані зашифровані за допомогою алгоритму, вказаного в заголовку.

При створенні токена на сервері він кодується за допомогою ключа, при отриманні він проводить таку ж операцію та перевіряє підпис на дійсність. Схему авторизації користувача наведено на рис. 8.

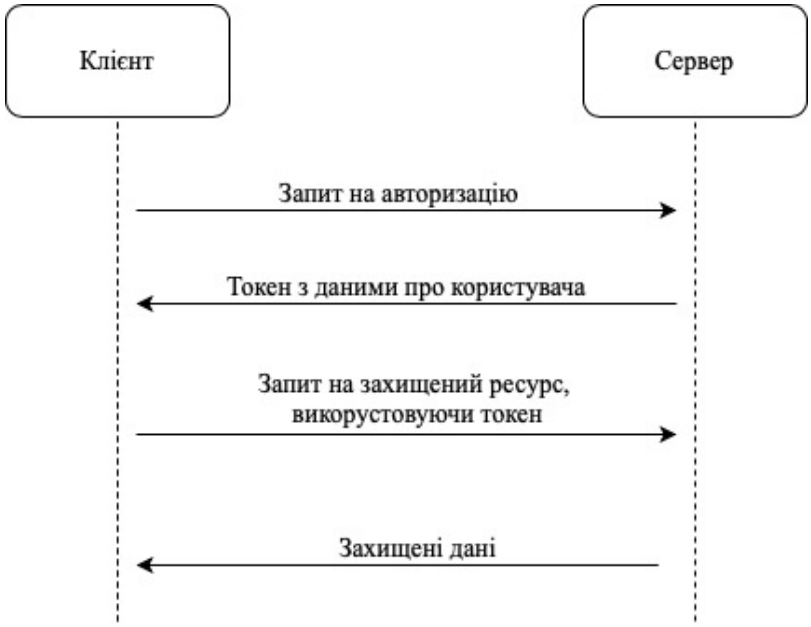



Рис. 8. Авторизація за стандартом JWT

### 4.3. Особливості реалізації авторизації клієнта

Додаток користувача авторизує клієнта за допомогою HTTP API додатку, який переадресує клієнта на сторінку соціальної мережі, задля надання делегованого доступу до його даних, таких як контакти або друзі, персональна інформація. Авторизація користувача в соціальній мережі відбувається за протоколом OAuth 2.0.

OAuth 2.0 – це відкритий стандартний протокол авторизації або фреймворк, який описує, як непов’язані сервери та сервіси можуть безпечно дозволити авторизований доступ до своїх ресурсів, фактично не надаючи дані для входу, такі як пароль та логін. Мовою авторизації цей протокол надає сторонню, делеговану авторизацію. Схему авторизації згідно протоколу OAuth 2.0 наведено на рис. 9. Переваги протоколу:

- безпечність – користувач надає доступ власноруч;
- серверу не доступні пароль та логін користувача.

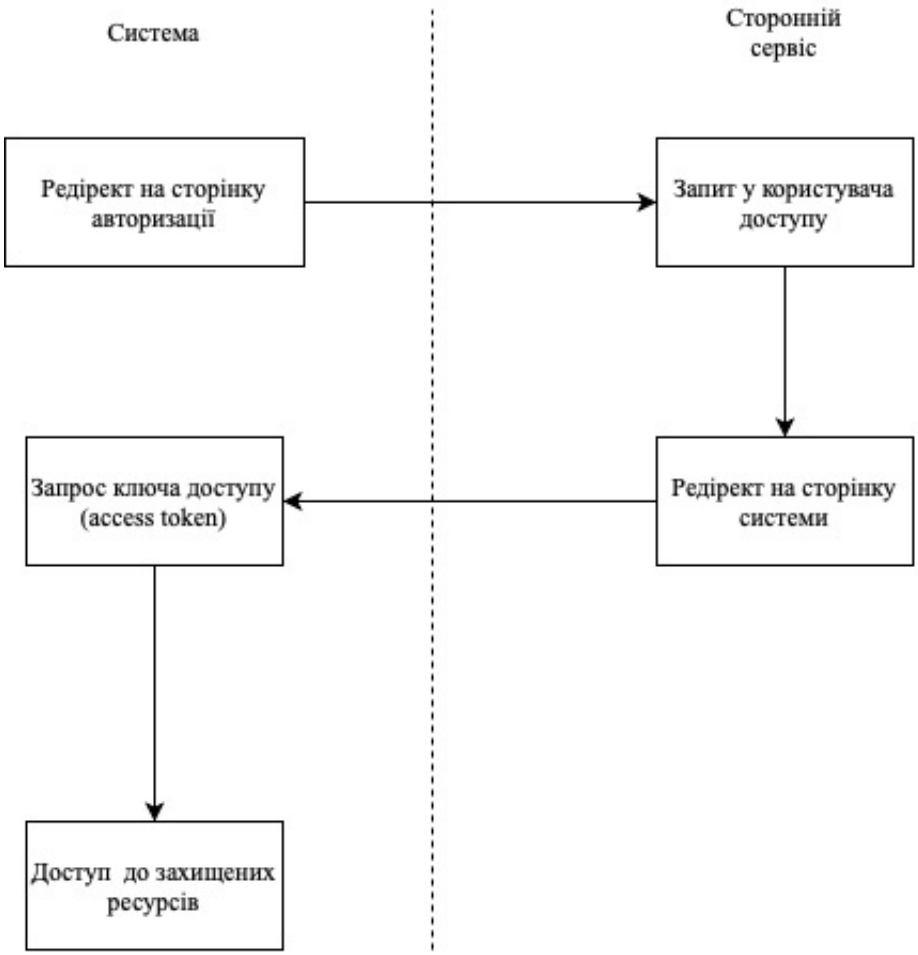



Рис. 9. Авторизація згідно протоколу OAuth 2.0

## 4.4. Тестування програмного забезпечення

Димове тестування – вид тестування програмного забезпечення, що перевіряє функціонування основних компонентів на відповідність основним сценаріям взаємодії. Зазвичай використовується для верифікації конкретної версії програмного забезпечення, задля виявлення очевидних помилок і для подальшого економії часу та ресурсів.

Тестування системи проводилось відповідно до вимог описаних в підрозділі 3.2, тому було сформовано набори тестових випадків (або тест-кейсів) задля проведення димового тестування, що наведені у таблицях 4.1 – 4.3.

Таблиця 4.1

Тест-кейси перевірки статусу системи

| № | Мета | Опис дій | Очікування |
|---|---|---|---|
| 1 | Перевірити можливість компіляції додатку в JavaScript. | Запуск скрипта компіляції | TypeScript або клієнтський JavaScript код скомпільований в JavaScript код без помилок |
| 2 | Перевірити початкову можливість запуску проєкту | Ізольований запуск компонентів системи | Всі компоненти системи запустились без помилок і попереджень |

Таблиця 4.2

Тест-кейси перевірки авторизації

| № | Мета | Опис дій | Очікування |
|---|---|---|---|
| 1 | Перевірити авторизацію користувача за допомогою логіна і пароля | Запит на отримання токену доступу | Отримання токена доступу і переадресація на персональний кабінет |
| 2 | Перевірити авторизацію користувача за допомогою ключа доступу до HTTP API | Запит на захищений ресурс HTTP API сервера | Отримання відповіді без помилки авторизації |
| 3 | Перевірити авторизацію клієнта через соціальні мережі | Запит на HTTP API на ресурс авторизації клієнта | Переадресація клієнта на соціальну мережу з подальшим отриманням токена делегованого доступа |

Таблиця 4.3

Тест-кейси перевірки надання рекомендацій

| № | Мета | Опис дій | Очікування |
|---|---|---|---|
| 1 | Перевірити створення дії клієнта | Запит на створення дії клієнта | Отримання статуса створення та створену дії |
| 2 | Перевірити коректність рекомендації конкретного для клієнта | Запит на отримання рекомендацій | Отримання рекомендації для клієнта що пов'язаний з іншим клієнтом |
| 3 | Перевірити авторизацію клієнта через соціальні мережі | Запит на HTTP API на ресурс авторизації клієнта | Переадресація клієнта на соціальну мережу з подальшим отриманням токена делегованого доступа |
| 4 | Перевірити отримання доступних критеріїв оцінювання | Запит на HTTP API на ресурс отримання критеріїв | Отримання списку доступних критеріїв оцінювання |

## 4.5. Рекомендації щодо розширення функціональних можливостей проєкта

Розроблений веб-сервіс є мінімально працюючим продуктом, що задовольняє критерії бакалаврського дипломного проєкту, але для виходу на повноцінний ринок з можливістю подальшої монетизації розроблених функцій недостатньо. Запити користувачів швидко зростають, а конкуренти працюють над покращенням своїх продуктів, тому розроблений програмний додаток можна вдосконалити в наступних версіях продукту.

Також, при тестуванні системи фокус-групою було отримано ряд рекомендацій для покращення системи, які можна додати в наступні версії:

- реалізація відкритих програмних модулів для полегшення інтеграції рекомендаційної системи;
- можливість фільтрації та сортування статистичних даних;
- можливість авторизувати клієнта за допомогою декількох соціальних мереж;
- можливість отримувати статистичні дані за допомогою HTTP API;
- можливість змінювати пріоритет між характеристика користувача, а не тільки дій;
- можливість зв’язку з групою підтримкою системи;
- можливість придбати підписку на сервіс (монетизації системи);
- можливість отримувати рекомендації згідно типу товару;
- можливість встановлення двухфакторної авторизації;
- можливість зміни пароля;
- можливість відновлення пароля;
- необхідність верифікації електронної пошти;
- необхідність верифікації додатку користувача;
- можливість змінити електронну пошту.

Також, в розробленому програмному продукті наявні функції для подальшого покращення веб-сервісу (збереження додаткових даних про клієнтів, підтримка односторонніх зв’язків між клієнтами), що стверджує про те, що цей програмний продукт створений с перспективою подальшого поліпшення.

# ВИСНОВКИ

Метою виконаного дипломного проєкту є розроблення соціально-аналітичного веб-сервісу для побудови рекомендацій, а саме системи що надасть користувачам можливість інтегрувати систему рекомендацій в їх додатки.

Згідно результатам дослідження функціональності існуючих аналогів було сформульовано вимоги до розроблюваної системи. Проведений аналіз показав, що система поєднує в собі сучасні практики надання рекомендацій та адаптує їх для аналізу соціальних зв’язків користувачів.

Дослідження існуючих засобів, технологій, та фреймворків виявив доцільність використання JavaScript та TypeScript в якості мов програмування, NodeJS в поєднанні з NestJS в якості фреймворків для серверної частини проєкту, та бібліотеки React та Redux для розробки клієнтської частини проєкту.

Розроблення система надає можливість користувачам:

- авторизувати клієнтів в соціальних мережах;
- отримувати персоналізовані рекомендацій;
- переглядати статистику щодо роботи своїх додатків;
- налаштовувати пріоритет дій клієнтів, задля точності надання рекомендацій.

Розробку було виконано в повному обсязі згідно з технічним завданням, проведено тестування у відповідності до затверджених методик та програм тестування.

Використання даної системи дозволить користувачам рекомендувати своїм клієнтам товари, які, на основі їх соціальних зв’язків, можуть їх зацікавити.

## СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ


1. JavaScript: [Електронний ресурс]. Режим доступу: https://developer.mozilla.org/en-US/docs/Web/JavaScript
2. TypeScript Documentation: [Електронний ресурс]. Режим доступу: https://www.typescriptlang.org/docs/home.html
3. MongoDB Manual: [Електронний ресурс]. Режим доступу: https://docs.mongodb.com/manual/
4. Neo4j Documentation: [Електронний ресурс]. Режим доступу: https://neo4j.com/docs/
5. What Is Graph Database: [Електронний ресурс]. Режим доступу: https://aws.amazon.com/nosql/graph/
6. NestJS: [Електронний ресурс]. Режим доступу: https://docs.nestjs.com/fundamentals/custom-providers
7. gRPC Documentation: [Електронний ресурс]. Режим доступу: https://grpc.io/docs/what-is-grpc/introduction
8. Microservices patterns: [Електронний ресурс]. Режим доступу: https://microservices.io/patterns/microservices.html
9. What is OAuth? How the open authorization framework works: [Електронний ресурс]. Режим доступу: https://www.csoonline.com/article/3216404/what-is-oauth-how-the-open-authorization-framework-works.html
10. Introduction to recommender systems: [Електронний ресурс]. Режим доступу: https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15adax
11. Recommender Systems in Practice: [Електронний ресурс]. Режим доступу: https://towardsdatascience.com/recommender-systems-in-practice-cef9033bb23a
12. Шпаргалки по безопасности: JWT: [Електронний ресурс]. Режим доступу: https://habr.com/ru/company/acribia/blog/457090/

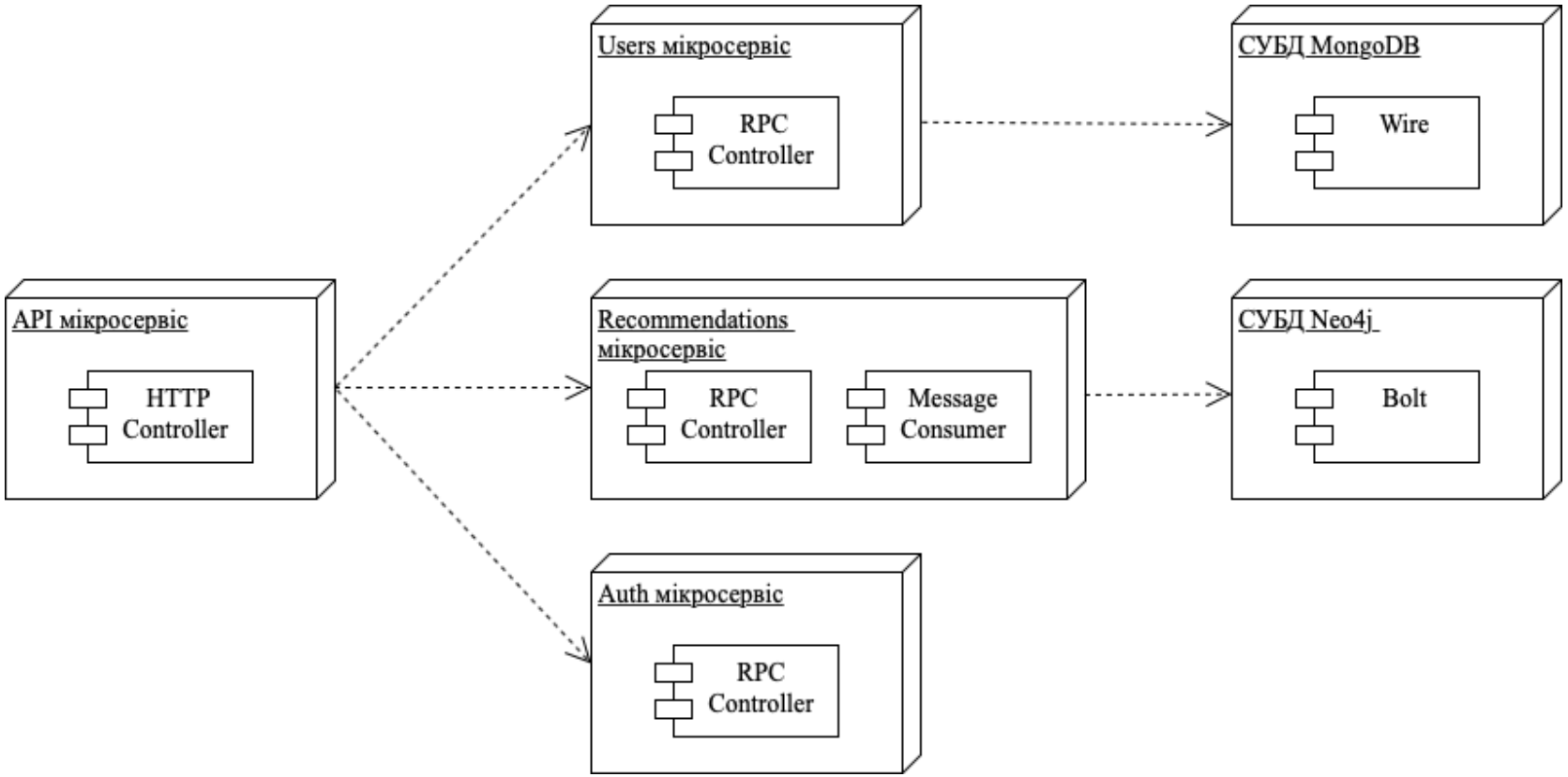
13. Cypher Query Language: [Електронний ресурс]. Режим доступу: https://riptutorial.com/neo4j/example/24410/cypher-query-language
14. Understanding and Using REST APIs: [Електронний ресурс]. Режим доступу: https://www.smashingmagazine.com/2018/01/understanding-using-rest-api/
15. Software Architecture Patterns — Layered Architecture: [Електронний ресурс]. Режим доступу: https://medium.com/@priyalwalpita/software-architecture-patterns-layered-architecture-a3b89b71a057
16. NodeJS Documentation: [Електронний ресурс]. Режим доступу: https://nodejs.org/en/docs/
17. React Documentation: [Електронний ресурс]. Режим доступу: https://reactjs.org/docs/getting-started.html
18. Misconceptions about Monorepos: Monorepo != Monolith: [Електронний ресурс]. Режим доступу: https://blog.nrwl.io/misconceptions-about-monorepos-monorepo-monolith-df1250d4b03c
19. Lerna: [Електронний ресурс]. Режим доступу: https://github.com/lerna/lerna
20. Babel Documentation: [Електронний ресурс]. Режим доступу: https://babeljs.io/docs/en/
21. How to securely store JWT tokens.: [Електронний ресурс]. Режим доступу: https://dev.to/gkoniaris/how-to-securely-store-jwt-tokens-51cf
22. RabbitMQ Documentation: Table of Contents: [Електронний ресурс]. Режим доступу: https://www.rabbitmq.com/documentation.html
23. Getting Started with WebStorm: [Електронний ресурс]. Режим доступу: https://www.jetbrains.com/help/webstorm/getting-started-with-webstorm.html
24. Best practices for using Git: [Електронний ресурс]. Режим доступу: https://deepsource.io/blog/git-best-practices/
25. Configuring ESLint: [Електронний ресурс]. Режим доступу: https://eslint.org/docs/user-guide/configuring

26. Building JavaScript Microservices with Node.js: [Електронний ресурс]. Режим доступу: https://www.twilio.com/blog/building-javascript-microservices-node-js
27. Write Your First Microservice in Nest JS: [Електронний ресурс]. Режим доступу: https://eslint.org/docs/user-guide/configuring
28. Building authentication for microservices using NestJS: [Електронний ресурс]. Режим доступу: https://dev.to/alesanchez/building-authentication-for-microservices-using-nestjs-1fne
29. OAuth 2.0: [Електронний ресурс]. Режим доступу: https://oauth.net/articles/
30. Presentational and Container Components: [Електронний ресурс]. Режим доступу: https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0
31. React Patterns: [Електронний ресурс]. Режим доступу: https://reactpatterns.com
32. Redux Form: [Електронний ресурс]. Режим доступу: https://redux-form.com/8.3.0
33. Design Patterns in TypeScript: [Електронний ресурс]. Режим доступу: https://refactoring.guru/design-patterns/typescript

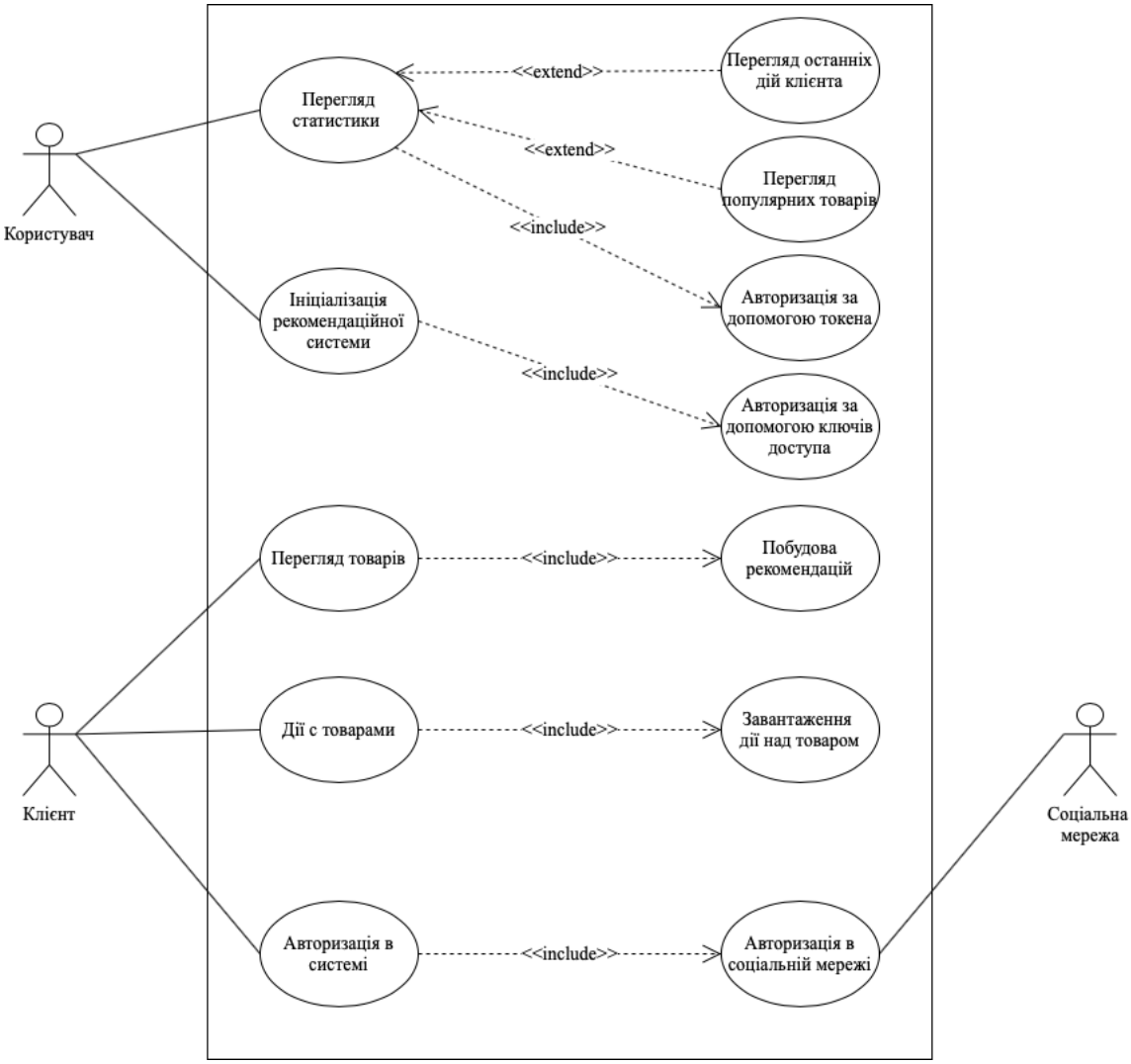
# ДОДАТКИ

**Додаток 1**

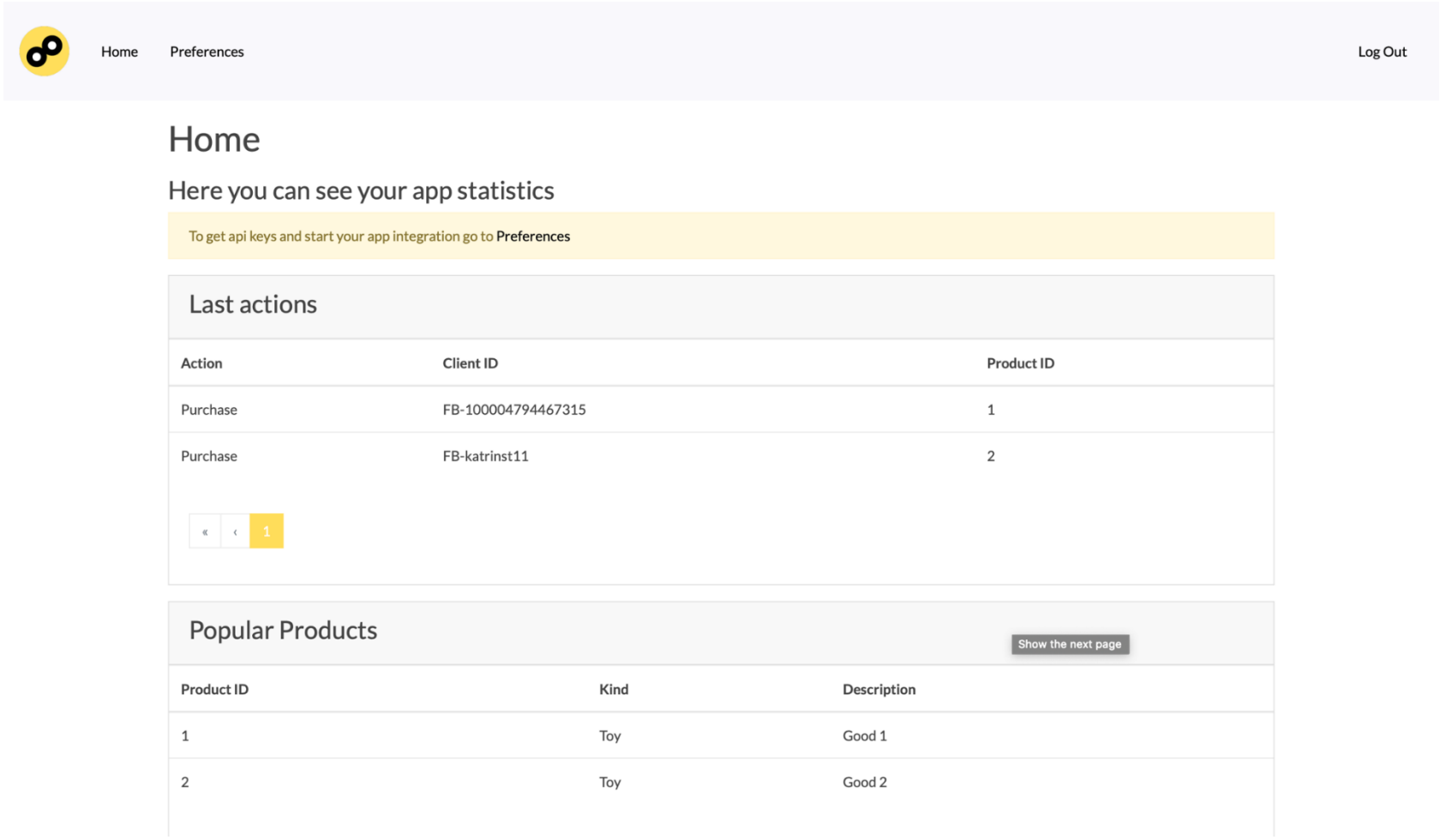
**Копії графічних матеріалів**



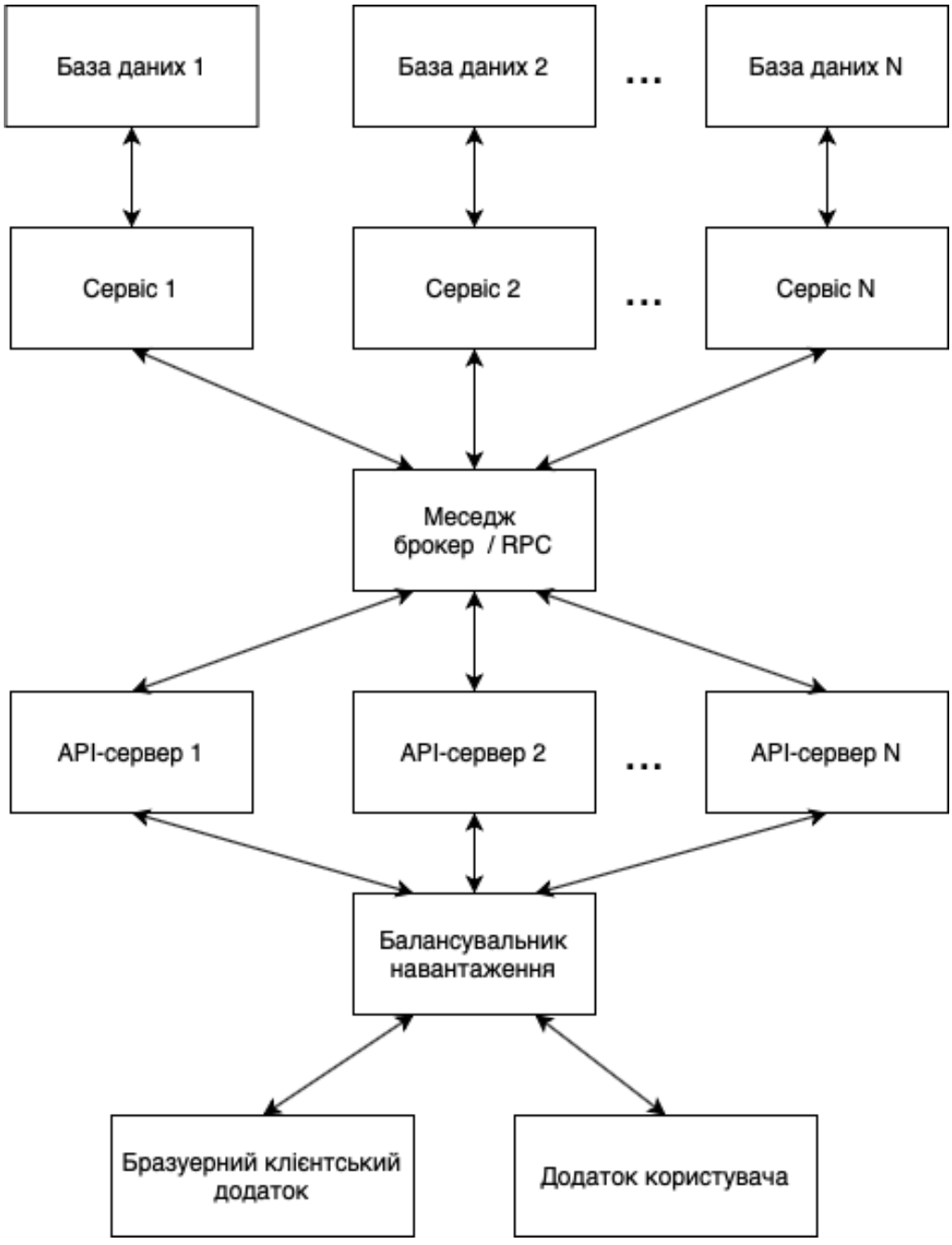

ДП.045440-06-99.
Соціально-аналітичний веб-сервіс для побудови рекомендацій. Діаграма компонентів.
UML діаграма




ДП.045440-07-99.
Соціально-аналітичний веб-сервіс для побудови рекомендацій. Діаграма варіантів використання. UML-діаграма прецедентів




Пінтак Дмитро Львович
гр. КП-62




Пінтак Дмитро Львович
гр. КП-62

# Додаток 2

# Текст програми

## 2.1. Модуль JWT

```
import {
  decode as decodeJWT,
  verify as verifyJWT,
  sign as signJWT,
  DecodedSign,
  Algorithm
} from 'jws'
import { Time } from '@meye/definitions'

type SignOpts = Partial<{
  algorithm: Algorithm
  timestamp: number
  expireIn: number
}>

export class JWTError extends Error {
  constructor(message: string) {
    super(message)
    this.message = message
  }
}

export class JWT {
  public readonly algorithm: Algorithm = 'HS256'
  public readonly expireIn = Time.Period.Millis.DAY /
Time.Period.Millis.SECOND

  constructor(private readonly secret: string, opts?: SignOpts)
{
    this.algorithm = opts?.algorithm ?? this.algorithm
    this.expireIn = opts?.expireIn ?? this.expireIn
  }

  static decode<T extends {} = {}>(
    token: string
  ): DecodedSign<T & { iat: number; exp: number }> | null {
    return decodeJWT<T & { iat: number; exp: number }>(token, {
json: true }) ?? null
  }

  static verify<T extends {} = {}>(token: string, secret:
string, algo: Algorithm): T {
    if (!verifyJWT(token, algo, secret)) throw new
JWTError('Token is not verified.')

    const decoded = JWT.decode<T>(token)
    if (!decoded?.payload) throw new JWTError('Token payload is
null.')
    if ((decoded.payload.iat + decoded.payload.exp) *
Time.Period.Millis.SECOND - Date.now() < 0)
      throw new JWTError('Token already expired.')
```

```
    return decoded.payload
  }

  static sign(tokenPayload: {}, secret: string, opts: SignOpts =
{}): string {
    const {
      algorithm = 'none',
      expireIn,
      timestamp = Date.now() / Time.Period.Millis.SECOND
    } = opts

    const header = { typ: 'JWT', alg: algorithm }
    const payload = { iat: Math.floor(timestamp), exp:
Math.floor(expireIn) }

    return signJWT({ secret, header, payload: { ...tokenPayload,
...payload } })
  }

  async decode<T extends {} = {}>(token: string):
Promise<DecodedSign<T> | null> {
    return JWT.decode<T>(token)
  }

  async verify<T extends {} = {}>(token: string): Promise<T> {
    return JWT.verify<T>(token, this.secret, this.algorithm)
  }

  async sign(payload: {}, opts: Exclude<SignOpts, 'algorithm'> =
{}): Promise<string> {
    return JWT.sign(payload, this.secret, {
      algorithm: this.algorithm,
      expireIn: this.expireIn,
      ...opts
    })
  }
}
```

## 2.2. Компонент «App.js»

```
import React from 'react'
import { Route, BrowserRouter, Switch } from 'react-router-dom'

import { DashboardPage, SignUpPage, LoginPage, ShopPage } from
'./pages'
import { AuthLayout, DashboardLayout } from './layouts'
import { PreferencesPage } from './pages/preferences'

function App() {
  return (
    <BrowserRouter>
      <Switch>
```

```
        <Route exact path="/dashboard">
          <DashboardLayout>
            <DashboardPage />
          </DashboardLayout>
        </Route>
        <Route exact path="/dashboard/preferences">
          <DashboardLayout>
            <PreferencesPage />
          </DashboardLayout>
        </Route>
        <Route exact path="/login">
          <AuthLayout>
            <LoginPage />
          </AuthLayout>
        </Route>
        <Route exact path="/sign-up">
          <AuthLayout>
            <SignUpPage />
          </AuthLayout>
        </Route>
        <Route exact path="/shop">
          <ShopPage />
        </Route>
      </Switch>
    </BrowserRouter>
  )
}

export default App
```

### 2.3. Схема сутності «User»

```
import {
  prop,
  plugin,
  arrayProp,
  modelOptions,
  getModelForClass,
  DocumentType,
  ReturnModelType
} from '@typegoose/typegoose'
import {
  ASCIIUniqueValueGenerator,
  UniquePlugin,
  UniquePluginModel
} from '@atlas/mongoose-unique-plugin'
import { Types, Aggregate, Error } from 'mongoose'

@modelOptions({
  schemaOptions: {
    minimize: false,
    toObject: { getters: true, virtuals: true, minimize: false
},
```

```
    toJSON: { getters: true, virtuals: true, minimize: false },
    collection: 'users'
  }
})
@plugin(new UniquePlugin({ fields: ['email'] }).plugin())
export class User {
  @prop({ required: true })
  email: string

  @prop({ required: true })
  password: string

  @prop({ required: true })
  oauthCallback: string
}

export const UserModel = getModelForClass(User) as
ReturnModelType<typeof User> &
  UniquePluginModel<DocumentType<User>> &
  PaginatePluginModel<DocumentType<User>>

export type UserDocument = DocumentType<User>
```

**Додаток 3**

**Копія презентації**

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

# Соціально-аналітичний веб-сервіс для побудови рекомендацій

Виконав: Пінтак Дмитро Львович

Керівник: Старший викладач кафедри ПЗКС, к.т.н. Рибачок Н. А

Київ – 2020

# ПОСТАНОВКА ЗАДАЧІ

**Мета проекту:** розробити соціально-аналітичний веб-сервіс для побудови рекомендацій

**Завдання:**

1. Проаналізувати існуючі рішення.
2. Обрати програмні інструменти
3. Розробити архітектуру та реалізувати додаток у вигляді веб-сервісу.
4. Протестувати розроблений додаток

# АКТУАЛЬНІСТЬ

Інтеграція системи в існуючі користувацькі додатки

- Збільшення прибутку компаній
- Допомога клієнтам знайти потрібне

# ІСНУЮЧІ АНАЛОГИ

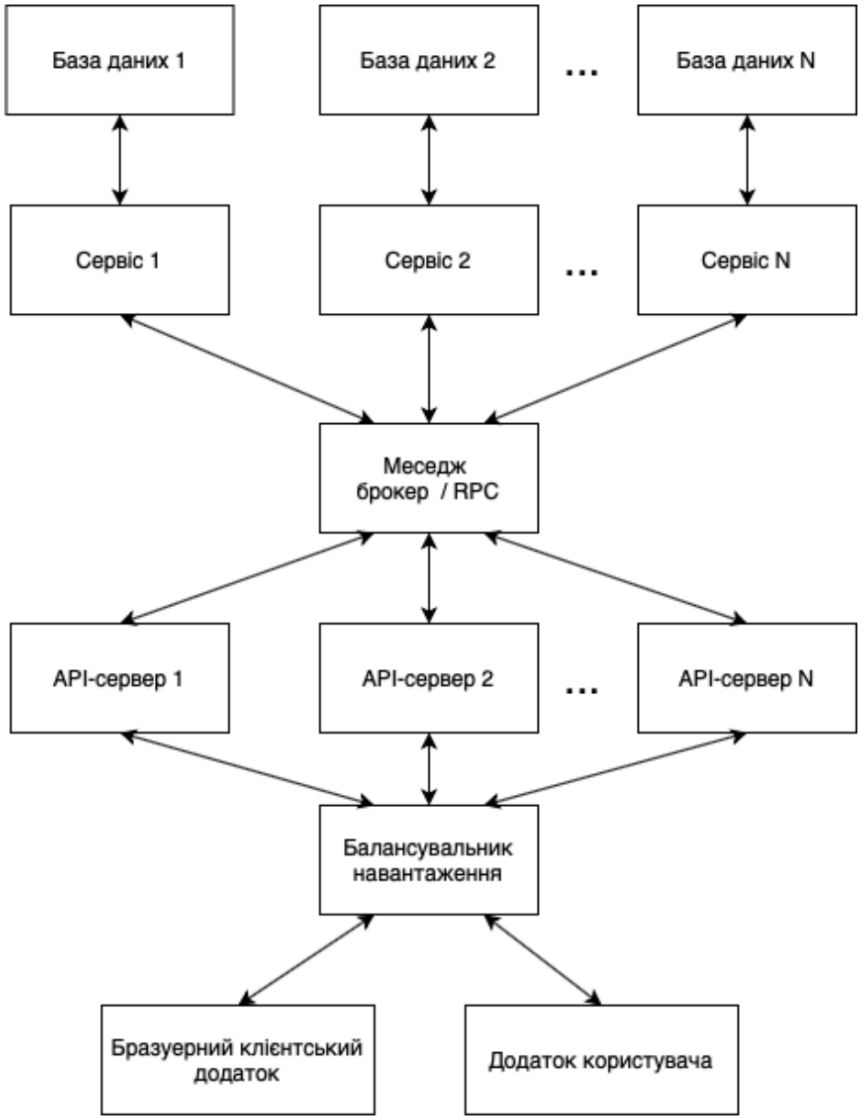
- Google Recommendation AI
- QuarticON


# РОЗРОБКА ВИМОГ

- Авторизація клієнта через соціальні мережі
- Перегляд журналу подій клієнтів
- Завантаження подій клієнтів та товарів в систему
- Отримання персоналізованих рекомендацій

# ЗАГАЛЬНА АРХІТЕКТУРА

1. Сервіс авторизації
2. Сервіс користувачів
3. Сервіс рекомендацій
4. Сервіс API



# ЗАСОБИ РОЗРОБЛЕННЯ. СУБД

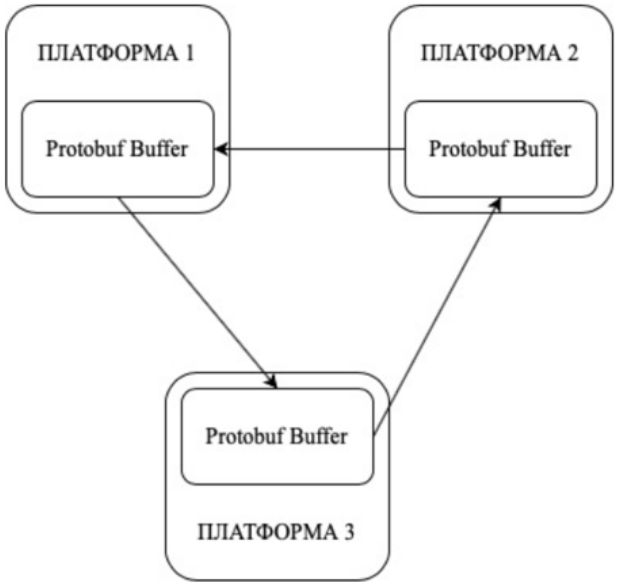

- SQL або NoSQL
- Графові бази даних
- Neo4j
- MongoDB


# ЗАСОБИ РОЗРОБЛЕННЯ. ФРЕЙМВОРКИ ТА БІБЛІОТЕКИ

- Мова програмування
- Клієнт
- Сервер



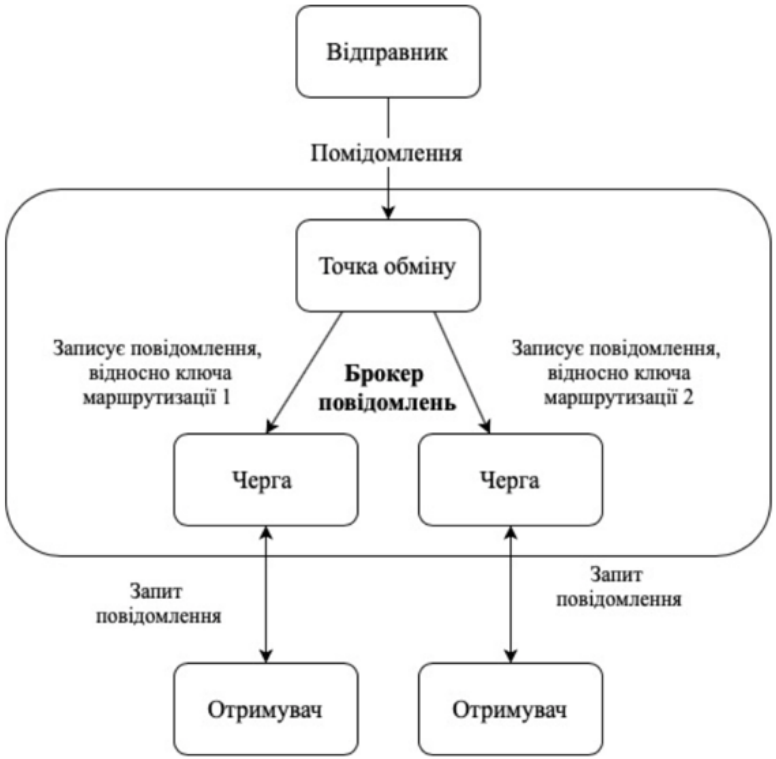

# ЗАСОБИ РОЗРОБЛЕННЯ. КОМУНІКАЦІЯ. gRPC




```proto
syntax = "proto3";

package users;

service UsersService {
  rpc GetById (GetByIdRequest) returns (User) {}

  rpc Create (CreateRequest) returns (User) {}
}

message GetByIdRequest {
  string id = 1;
}

message CreateRequest {
  string email = 1;
  string password = 2;
  string apiKey = 3;
}

message User {
  string email = 1;
  string apiKey = 3;
}
```

gRPC

# ЗАСОБИ РОЗРОБЛЕННЯ. КОМУНІКАЦІЯ. RabbitMQ / amqp



# РОЗРОБЛЕНІ АЛГОРИТМИ. НАДАННЯ РЕКОМЕНДАЦІЙ


Ролі:

- Клієнт
- Користувач
- Користувач-розробник

Структури даних:

- Рекомендація або виборка товарів
- Дія клієнта
- Товар

**Послідовність дій для отримання рекомендацій**:

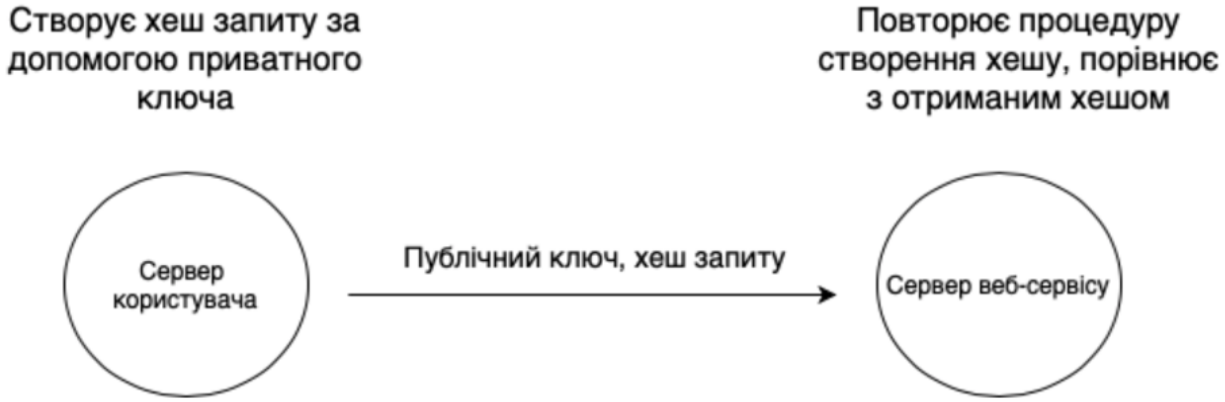
1. Реєстрація, авторизація користувача
2. Отримання ключів доступу до HTTP API

**Послідовність дій для отримання рекомендацій**:

1. Авторизація клієнта
2. Створення дій клієнта
3. Отримання персоналізованих рекомендацій

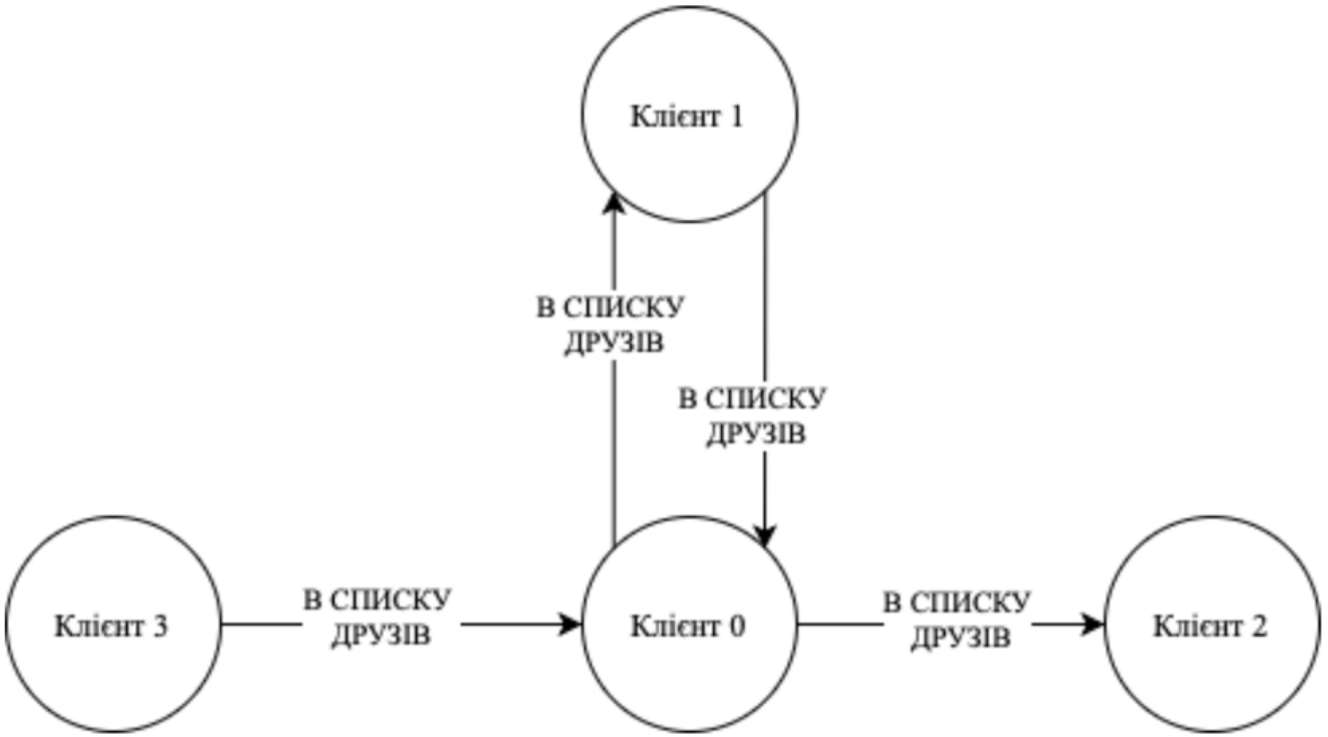
# РОЗРОБЛЕНІ АЛГОРИТМИ. ПІДПИС ЗАПИТІВ.

- Пара ключів
- HMAC
- Підпис запиту

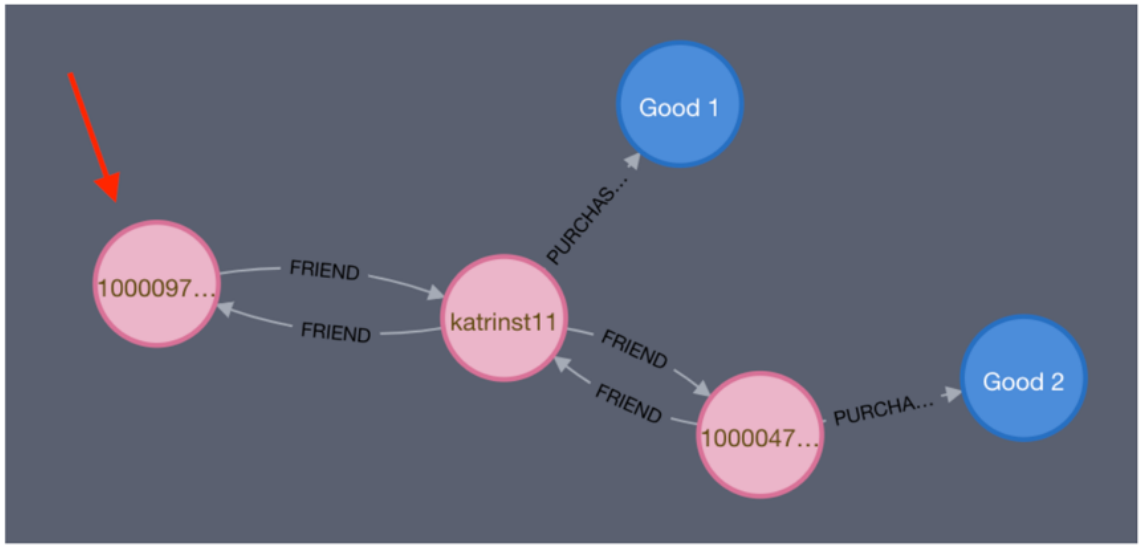


# РОЗРОБЛЕНІ АЛГОРИТМИ. НАДАННЯ РЕКОМЕНДАЦІЙ. АВТОРИЗАЦІЯ КЛІЄНТА. OAuth

# РОЗРОБЛЕНІ АЛГОРИТМИ. НАДАННЯ РЕКОМЕНДАЦІЙ. ПОБУДОВА ЗВ’ЗКІВ



# РОЗРОБЛЕНІ АЛГОРИТМИ. НАДАННЯ РЕКОМЕНДАЦІЙ. ОТРИМАННЯ РЕКОМЕНДАЦІЙ. Cypher





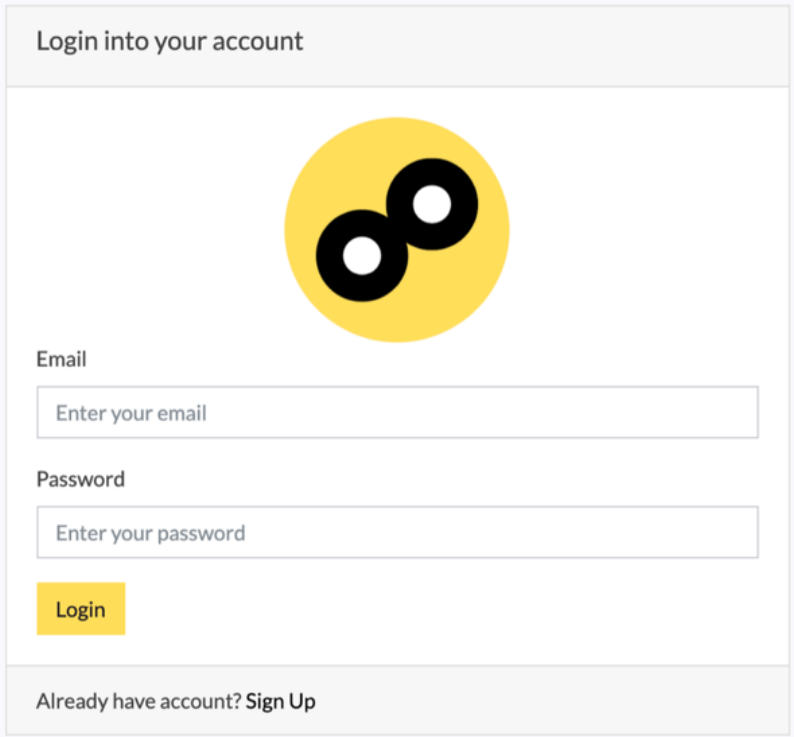
```
[{
  "description": "Good 1",
  "kind": "toy",
  "id": 1
},
{
  "description": "Good 2",
  "kind": "toy",
  "id": 2
}]
```


```
MATCH (c:Client {socialNetworkID: $clientID})-
[f:FRIEND * 1..]→(n)-[:PURCHASED]→(p: Product)
RETURN p, length(f) AS depth
ORDER BY depth
```

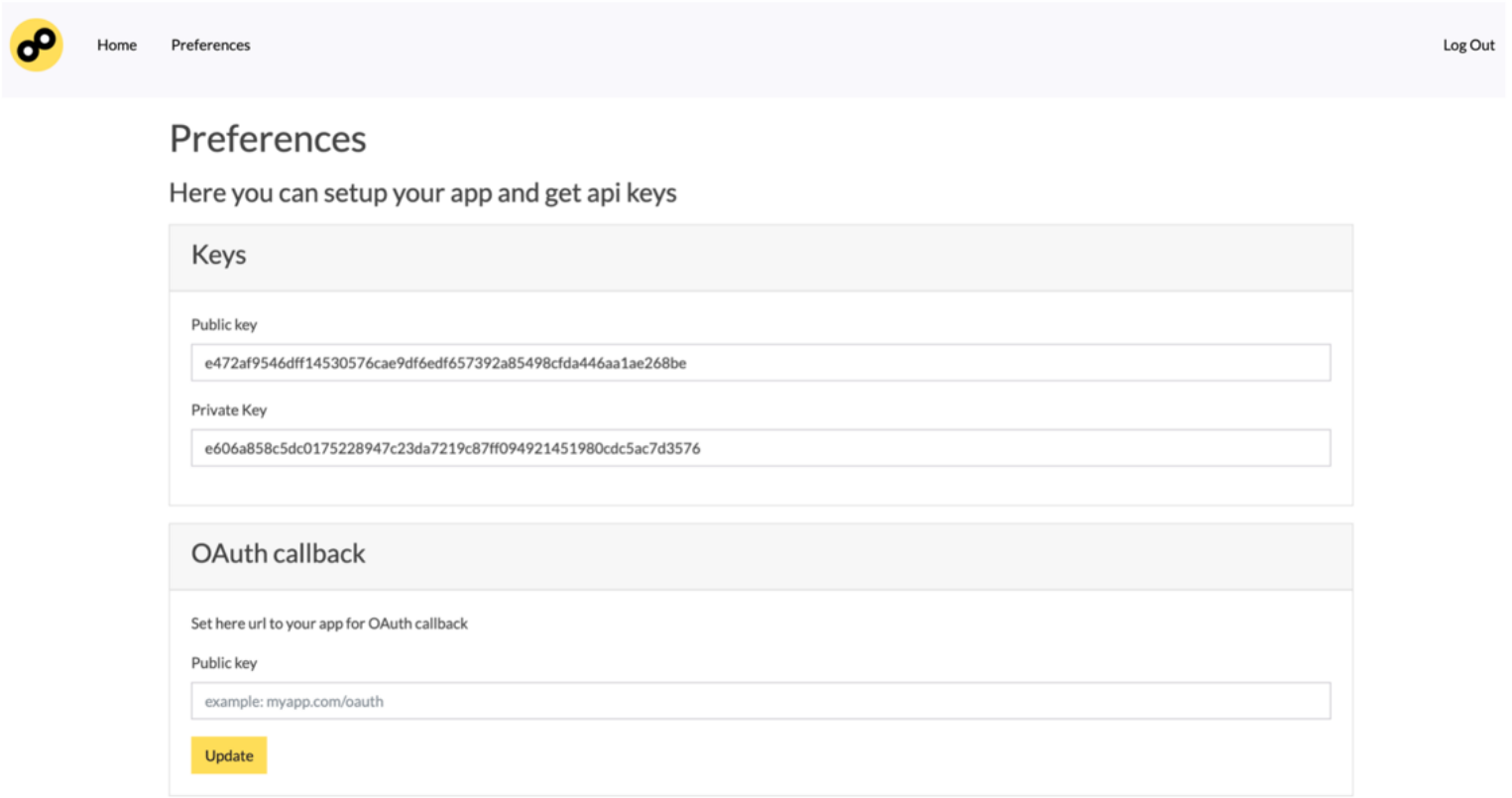
# РОЗРОБЛЕНІ АЛГОРИТМИ. НАДАННЯ РЕКОМЕНДАЦІЙ. ОТРИМАННЯ РЕКОМЕНДАЦІЙ. HTTP API

| ЗАПИТ | ПРИЗНАЧЕННЯ | ВІДПОВІДЬ |
|---|---|---|
| POST /clients/ action | Створення дії клієнта (придбав, вподобав) | Створена дія |
| POST /clients/ auth | Передаресація клієнта на соціальну мережу | Переадресація |
| GET / recommendations | Отримання рекомендацій | Виборка товарів |

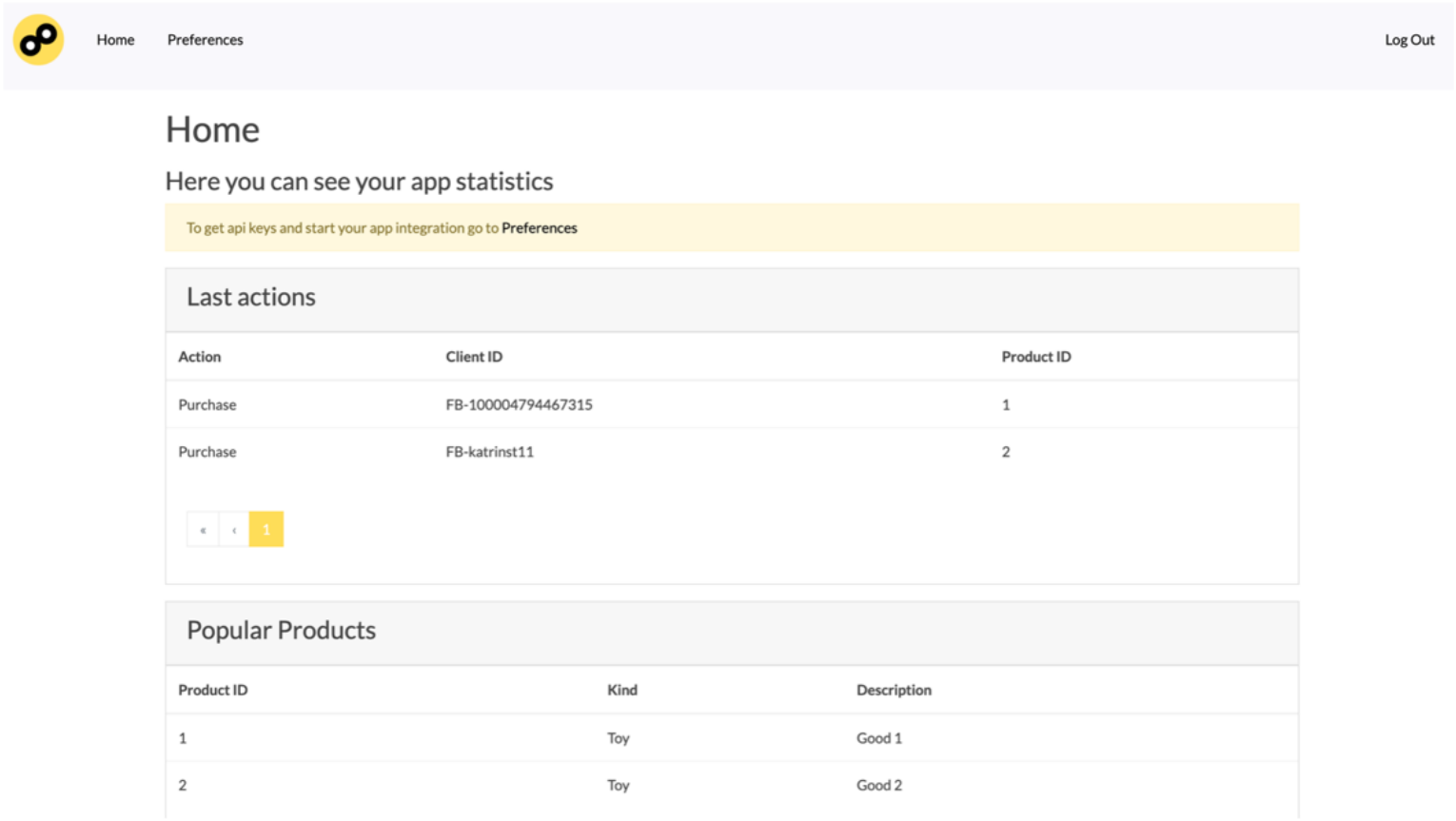
# АВТОРИЗАЦІЯ КОРИСТУВАЧА



# ОТРИМАННЯ КЛЮЧІВ ДОСТУПУ ДО HTTP API



# ПЕРЕГЛЯД СТАТИСТИКИ




# ТЕСТУВАННЯ

- тестування вручну згідно тест-кейсів
- smoke тестування
- автоматичне тестування

# ВИСНОВКИ

Виконані задачі процесу розбки ПЗ:

- Збір потреб та вимог
- Розроблення
- Тестування


# Дякую за увагу!

**Факультет прикладної математики**

**Кафедра програмного забезпечення комп'ютерних систем**

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2019 р.

# СОЦІАЛЬНО-АНАЛІТИЧНИЙ ВЕБ-СЕРВІС ДЛЯ ПОБУДОВИ РЕКОМЕНДАЦІЙ

## Програма та методика тестування

ДП.045440-04-51

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Наталя РИБАЧОК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Дмитро ПІНТАК

2019

**ЗМІСТ**

## 1. ОБ’ЄКТ ВИПРОБУВАНЬ

Соціально-аналітичний веб-сервіс для побудови рекомендацій, серверна частина якого розроблена з використанням фреймворку NestJS на основі платформи NodeJS, а клієнтська – з використанням фреймворку ReactJS.

## 2. МЕТА ТЕСТУВАННЯ

У процесі тестування має бути перевірено наступне:

1) відповідність функціональних можливостей, реалізованих у програмному застосунку до заявлених;
2) коректність поведінки програмного застосунку;
3) забезпечення належного рівня безпеки даних;
4) зручність роботи з веб-сервісом.

## 3. МЕТОДИ ТЕСТУВАННЯ

Тестування проводитиметься на рівні «системного тестування», тобто буде перевірено кожен модуль розробленого програмного продукту та зв’язки між ними. Для проведення тестування було обрано метод White Box Testing. При такому методі перевіряється як поведінка програмного продукту, так і код.

Використовуються наступні методи:

1) для проведення функціонального тестування Critical path test (тестування критичного шляху);
2) для тестування продуктивності програмного забезпечення load testing (навантажувальне тестування) та stress testing (стресс-тестування);
3) тестування інтерфейсу.

## 4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Коректність роботи веб-сервісу тестується з використанням таких методів та утиліт:

1) введенням недопустимих значень в поля, що можна редагувати;
2) ручного тестування відповідності реалізованих функціональних вимог до заявлених;
3) тестування при максимальному навантаженні;
4) тестування стабільності роботи при різних умовах;
5) тестування веб-сервісу в різних браузерах;
6) тестування зручності користувацького інтерфейсу опитуванням потенційних користувачів.

**Факультет прикладної математики**

**Кафедра програмного забезпечення комп'ютерних систем**

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2020 р.

# СОЦІАЛЬНО-АНАЛІТИЧНИЙ ВЕБ-СЕРВІС ДЛЯ ПОБУДОВИ РЕКОМЕНДАЦІЙ

## Керівництво користувача

ДП.045440-05-34

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Наталя РИБАЧОК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Дмитро ПІНТАК

2020

**ЗМІСТ**

## 1. Опис структури веб-сервісу

Веб-сервіс для аналізу та трекінгу задач складається з наступних сторінок:

- «Login»;
- «Sign Up»;
- «Home»;
- «Preferences».

Перехід між сторінками відбувається за натисканням відповідних кнопок на верхній навігаційній панелі або за посиланнями.

## 2. Процедура авторизації користувача

Сторінка «Login» відкривається за змовчуванням для неавторизованих користувачів (рис. 1). Сторінка містить форму для вводу необхідних для авторизації даних: електронної пошти та пароля. Після введення коректних даних та натиснення кнопки «LOGIN», користувач увійде в систему. Також, на сторінці міститься посилання на сторінку «Sign Up».

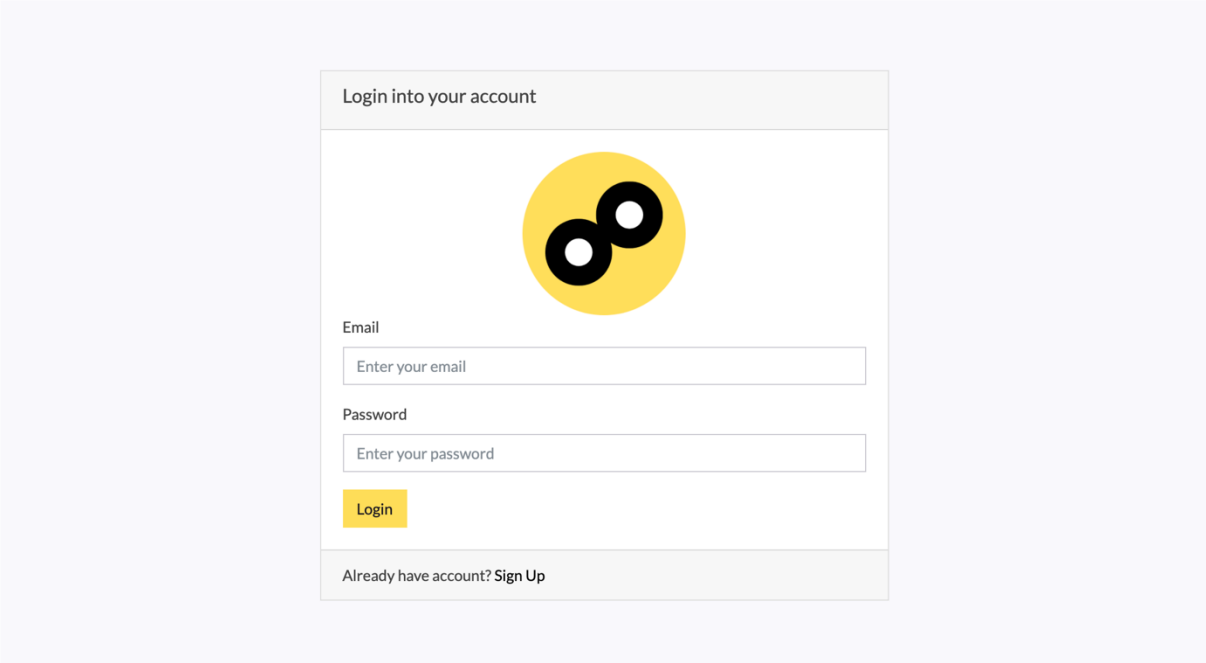



Рис. 1. Сторінка «Login»

## 3. Процедура реєстрації користувача

Сторінка «Sign Up» (рис. 2) містить форму для вводу необхідних для реєстрації даних: адресу поштової скриньки та паролю. Після введення коректних даних та натиснення кнопки «Sign Up», користувач увійде в систему. Також, на сторінці міститься посилання на сторінку «Login».

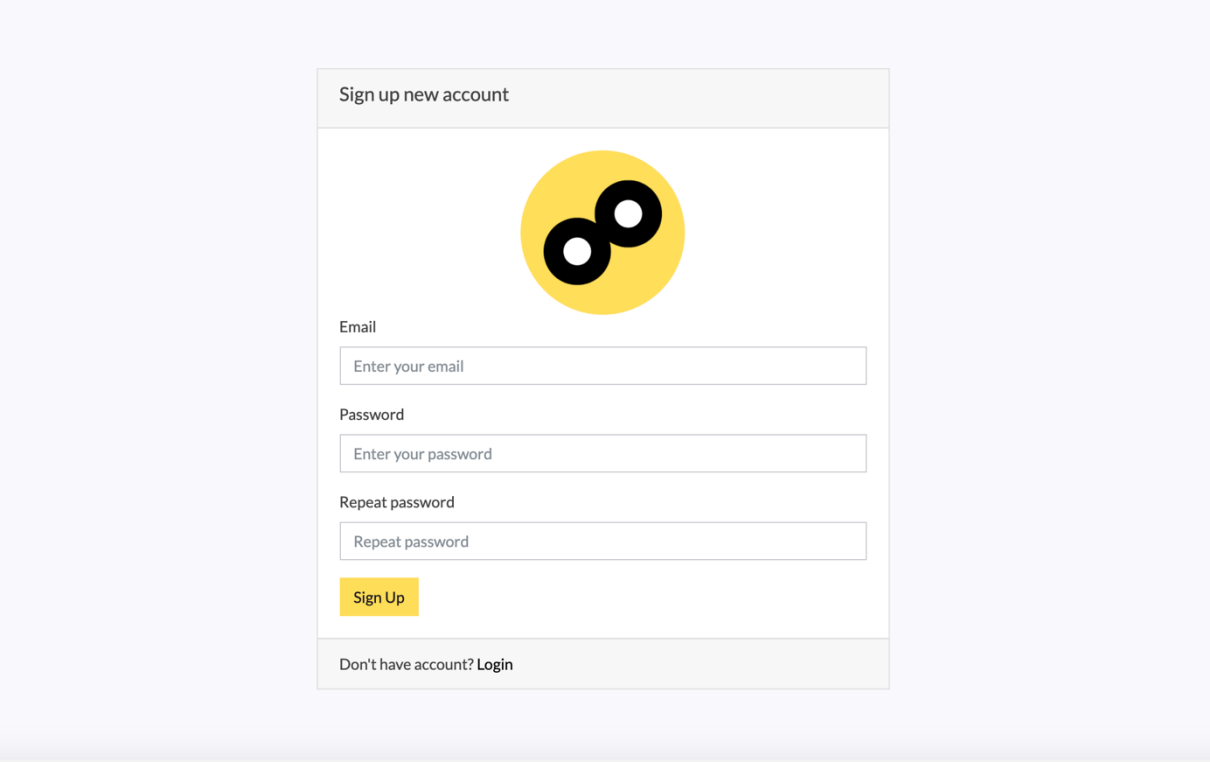



Рис. 2. Сторінка «Sign Up»

## 4. Сторінка «Home»

Після успішної авторизації, користувач потрапляє на сторінку «Home» (рис. 3). На цій сторінці користувач може переглянути останні дії клієнтів, на популярні товари, також користувач може перейти на сторінку «Preferences».

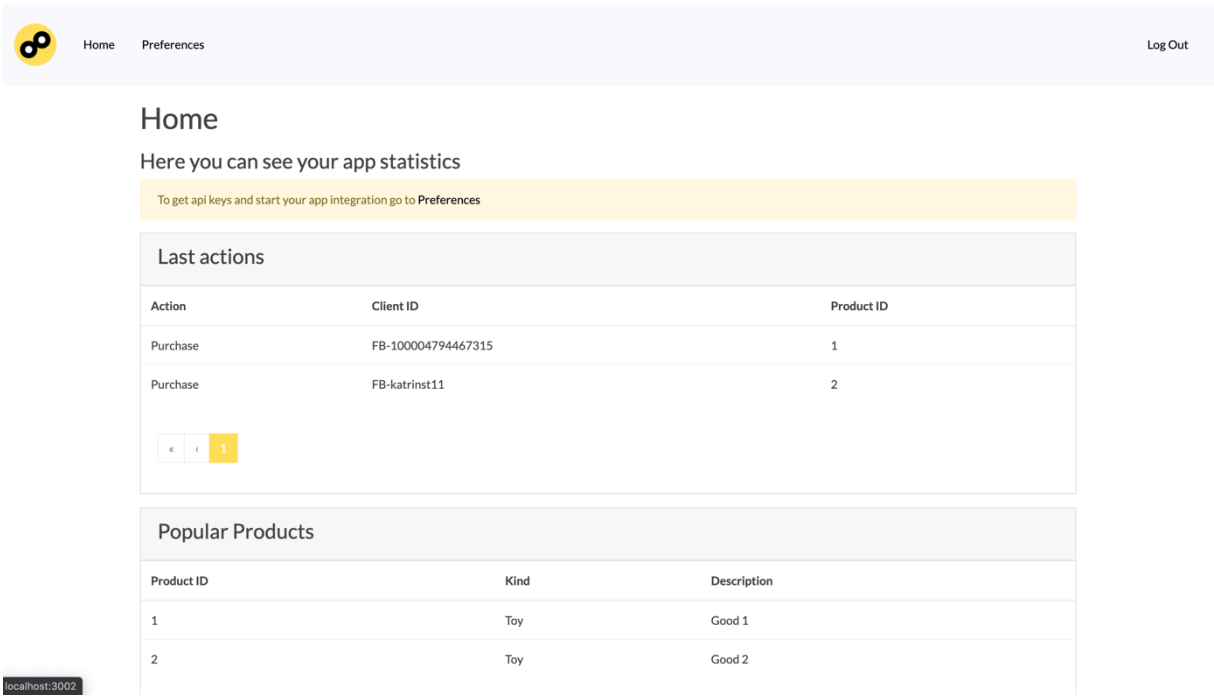



Рис. 3. Сторінка «Home»

## 5. Сторінка «Preferences»

Авторизований користувач може потрапити на сторінку «Preferences», використовуючи навігацію (рис. 4). На цій сторінці користувач може отримати ключі доступу до HTTP API (рис. 5). Також користувач може задати адрес переадресація OAuth авторизації (рис. 6).

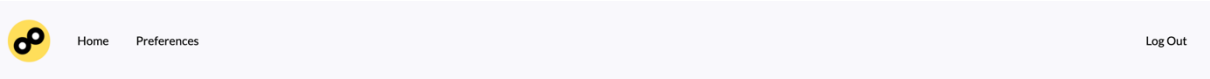



Рис. 4. Навігація

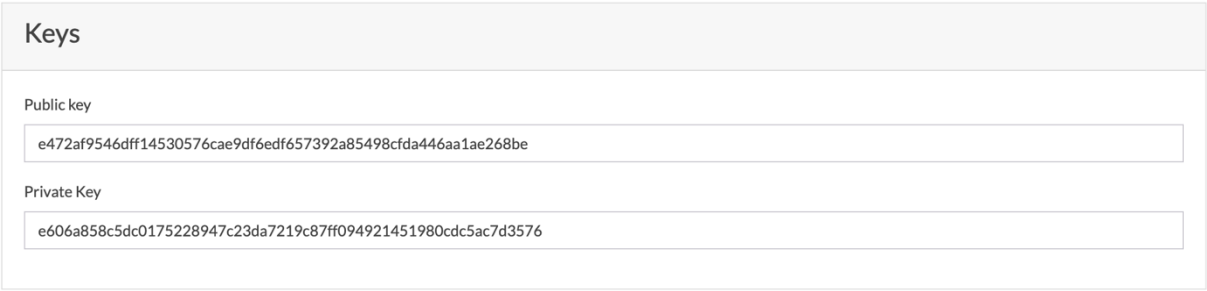



Рис. 5. Ключі доступу до HTTP API

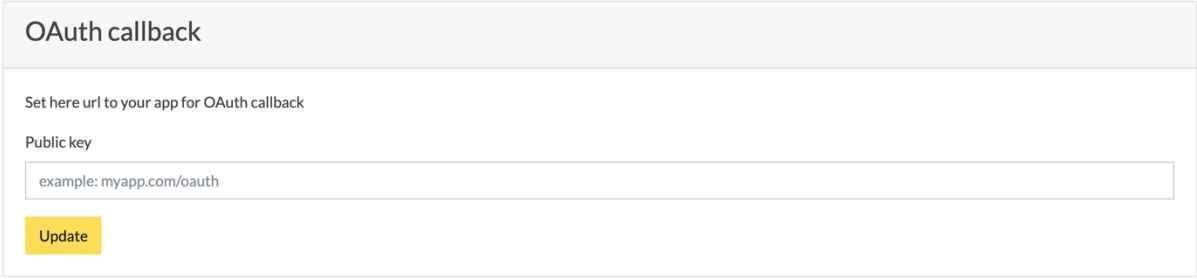



Рис. 6. Адрес OAuth переадресації

## 6. Вихід з акаунту

Після натиснення на праву кнопку горизонтальної навігаційної панелі відбувається вихід з акаунту (рис. 7).

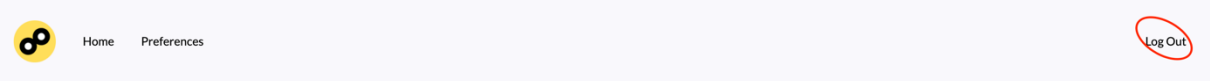



Рис. 7. Вихід з акаунту